Compréhension de Textes dans un domaine technique :le système ACTES.Application des Grammaires d'Unification et de la Théorie du Discours

Thierry Chanier

UNIVERSITE de PARIS XIII
Centre Scientifique et Polytechnique
Laboratoire d'Informatique de Paris Nord
Villetaneuse

THESE

Présentée pour obtenir le grade de

Docteur d'Université

Spécialité Informatique

par

Thierry CHANIER

Compréhension de Textes dans un domaine technique : le système ACTES. Application des Grammaires d'Unification et de la Théorie du Discours

Soutenue le 28 juin 1989 devant le jury :

MM.

Daniel COULON	Président
Daniel KAYSER	Directeur
Gérard SABAH	Rapporteur
Patrick SAINT-DIZIER	Rapporteur
Célestin SEDOGBO	

A Monique

REMERCIEMENTS

Merci à

Daniel Coulon pour avoir accepté la présidence de cette thèse et s'être interessé à mes travaux.

Gérard Sabbah et Patrick Saint-Dizier pour la confiance qu'ils m'ont témoignée en examinant ce travail.

Daniel Kayser pour avoir accepté de diriger cette thèse et m'avoir accordé sa confiance tout en me laissant maître de choix qu'il ne partageait pas forcément. Son soutien, ses conseils d'orientation m'ont aidé à traverser les moments de doute. Je lui suis gré de sa grande disponibilité et de sa rigueur scientifique.

Celestin Sedogbo pour m'avoir offert l'occasion de travailler dans le domaine du traitement automatique du langage naturel et avoir guidé mes premiers pas.

André Chamard pour son point de vue réfléchi sur la DRT. Il retrouvera dans certains passages de cette thèse l'écho de discussions passées.

Michel Chambreuil pour m'avoir communiqué sa passion pour l'Intelligence Artificielle. J'ai beaucoup apprécié sa façon d'aborder le problème du traitement du langage naturel, sans exclusive, en partageant son intérêt entre les approches formelles et celles dites "procédurales".

Corinne Fournier pour sa coopération chaleureuse dans le projet ACTES, pour son appréciation très fine des richesses du langage et des difficultés de son traitement.

Catherine Rodriguez pour son aide dans l'implantation du système ACTES et ses nombreux avis critiques et constructifs.

Toute l'équipe de l'ancienne division Intelligence Artificielle du Centre de Recherche BULL, pour leur enthousiasme, leur disponibilité de tous les instants, leurs intérêt éclectique pour toutes les disciplines du domaine.

Monique et Aurélie pour avoir accepté pendant 42 mois, les séparations hebdomadaires et mon manque de disponibilité. Sans leur soutien, cette thèse n'aurait pu voir le jour.

RESUME

Le traitement automatique de textes de spécifications écrits en langage naturel pose au moins deux problèmes : disposer d'outils généraux permettant de reconnaître les constituants de ces textes ; trouver une représentation de leur contenu permettant à l'utilisateur de les modifier et de vérifier leur cohérence. Les textes analysés ici spécifient les logiques d'alarmes d'un avion. On veut en extraire des règles pour un système expert simulant le comportement des processeurs embarqués, chargés de gérer ces alarmes. Le système ACTES offre, d'une part, un formalisme permettant de définir des grammaires et de construire une représentation sémantique intermédiaire des textes, et, d'autre part, un environnement de développement de grammaires et d'analyse de textes. Le formalisme s'inspire des grammaires d'unification et de la Théorie de la Représentation du Discours. L'approche mise en oeuvre ici présente l'avantage d'être unifiée: la syntaxe et la sémantique sont traitées par la même opération d'unification Prolog. Toutefois, Prolog ne prenant pas en compte les mécanismes d'héritage, une opération d'unification étendue gérant les contraintes hiérarchiques de types entre éléments du formalisme a été définie. L'environnement du système comprend, notamment, un module de résolution d'anaphores intégrant des sources de contraintes multiples ayant chacune leur propre stratégie dans une architecture en 'tableau noir'. Quant au modèle de représentation finale du contenu d'un texte, il s'inspire de la logique des défauts. Cette logique reflète bien la construction non-monotone de ces textes. Elle permet d'envisager la modification interactive avec vérification de cohérence.

MOTS CLES: Compréhension de textes en langage naturel, Grammaire d'unification, Représentation du discours, Résolution d'anaphores, Application de la Logique des défauts.

ABSTRACT

Automatic processing of specification texts written in natural language raises at least two problems: the availability of general tools to recognize the constituents of the texts; finding a way to represent their contents allowing the user to modify and check them for consistency. The texts analyzed here specified the alarm logic for an aircraft. The long-term aim is to extract rules for an expert system simulating the behavior of the airborne processors responsible for managing the alarms. The ACTES system offers, on the one hand, a formalism allowing grammars to be defined and an intermediate semantic representation of the texts to be made, and , on the other hand, an environment for the development of grammars and for texts analysis. The formalism is inspired from unification-based grammars and the Discourse Representation Theory. The approach implemented has the advantage of being unified: the syntax and the semantics are processed by the PROLOG unification operation. But since Prolog does not take into account hierarchical mecanisms, we defined an extended unification operation which manages hierarchical constraints among the typed elements of the formalism. The environment includes, among others, an anaphora resolver which integrates multiple constraint sources which have their own strategies in a blackboard architecture. As regards the final representation model of the content of a text, it is inspired from the Default Logic. This logic satisfactorily reflects the non-monotonic construction of these texts. It allows interactive modification with consistency checks to be envisaged.

KEYWORDS: Natural Language Text Understanding, Unification-based Grammars, Discourse Representation, Anaphora Resolution, Default Logic.

TABLE DES MATIERES

ABREVIATIONS

ATN	Augmented Transition Network
ACTES	Acquisition de Connaissances à partir de Textes pour un Expert en Spécifications
AVAG	Attribute VAlue Grammar
CUG	Categorial Unification Grammar
DAG	Directed Acyclic Graph
DCG	Definite Clause Grammar
Dref	Discourse Referent (référent du discours)
DRS	Discourse Representation Structure
DRT	Discourse Representation Theory
FUG	Functional Unification Grammar
GPSG	Generalized Phrase Structure Grammar
IA	Intelligence Artificielle
LFG	Lexical Functional Grammar
LN	Langage Naturel
PATR	PArsing and TRanslation
RC	Représentation des Connaissances
TLN	Traitement informatique du Langage Naturel
UCG	Unification Categorial Grammar

0. INTRODUCTION

0.1. Les données du problème

Les résultats présentés dans cette thèse sont l'aboutissement de travaux menés consécutivement suivant deux axes très différents.

Nos premières recherches dans le domaine du traitement du langage naturel ont concerné le développement d'outils généraux pour l'écriture de grammaires. L'objectif était d'intégrer dans un même formalisme certaines caractéristiques propres aux grammaires d'unification et une théorie sémantique sur la représentation du discours, la Théorie de la Représentation du Discours (DRT), en utilisant au mieux les possibilités du langage Prolog. Ces outils devaient permettre de réécrire de façon plus "déclarative" une grammaire syntaxique générale du français et de décrire des phénomènes liés à la traduction de certains quantificateurs du langage naturel et à quelques cas d'anaphores. Aucun domaine d'application particulier, aucun corpus précis n'était alors considéré.

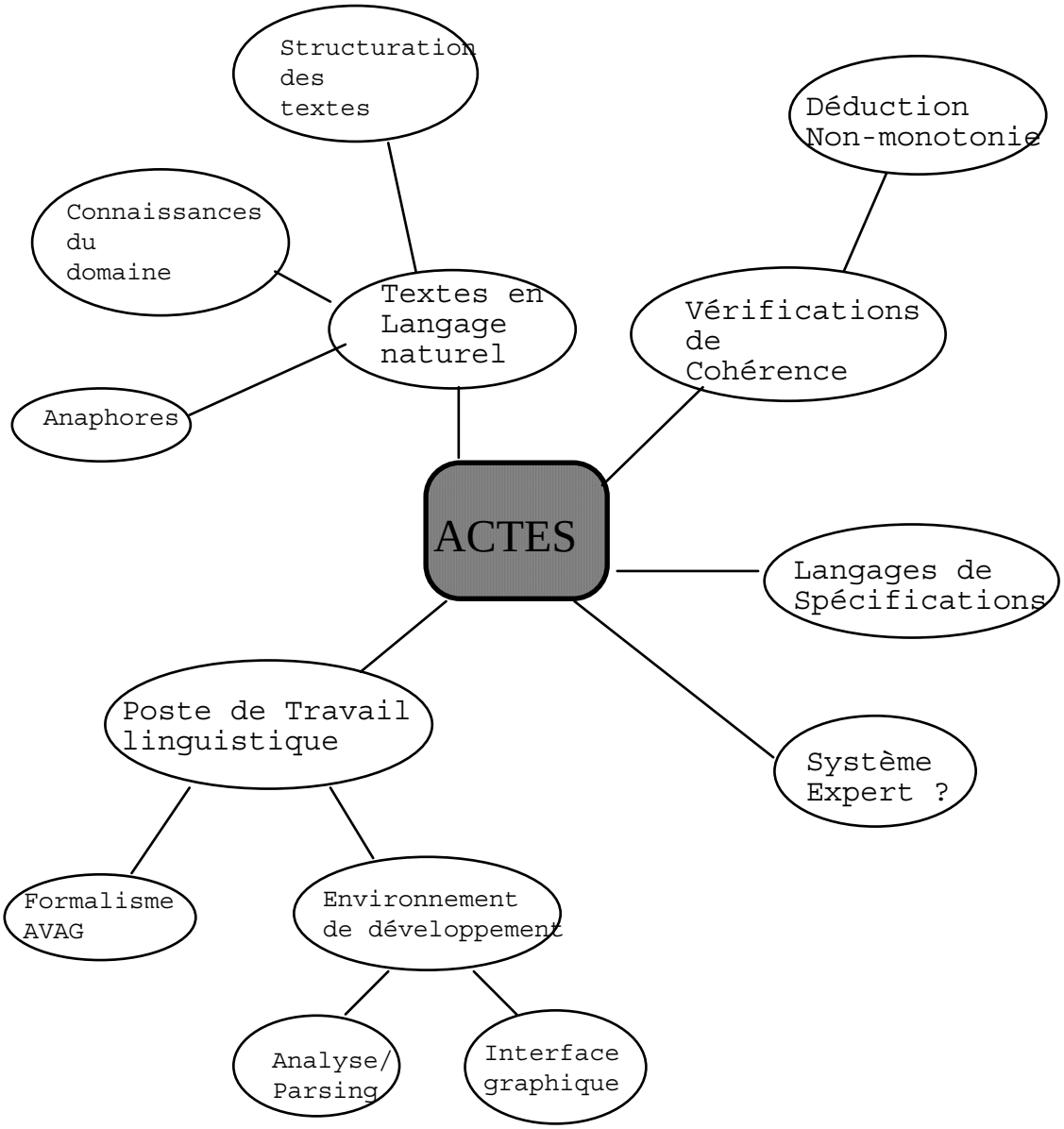
Le deuxième axe de travail, plus récent, le projet ACTES[1], s'est présenté de façon très différente, pour ne pas dire opposée, sous bien des aspects. Il s'agissait de réaliser un prototype permettant d'extraire automatiquement des connaissances à partir de textes techniques, écrits dans un sous-ensemble du langage naturel, spécifiant les déclenchements d'alarmes dans un avion.

Ces textes de spécification produits par des experts ont deux types d'utilisateurs: les programmeurs chargés d'implanter les procédures décrivant dans quelles conditions les processeurs embarqués sur un avion déclenchent les alarmes et avertissent le pilote ; les ingénieurs cogniticiens qui mettent au point un système expert simulant ces alarmes. Aujourd'hui les règles de production du système expert sont extraites "à la main" par ces ingénieurs. ACTES se présentait donc comme un projet prospectif sur l'automatisation du processus d'extraction des règles de production à partir de ces textes.

---

[1] ACTES (Acquisition de Connaissances à partir de Textes pour un Expert en Spécifications) est un projet commun Avions Marcel Dassault - Centre de Recherche BULL, partiellement financé par la Direction des Recherches Etudes Techniques (DRET) du Ministère de la Défense.

Les spécifications du projet insistaient sur deux types de fonctionnalités qui devaient guider l'élaboration du prototype: d'une part, la mise en place d'**outils généraux d'analyse de textes** ; d'autre part, la représentation de ces textes sous une forme permettant la mise-à-jour ainsi que des **vérifications de cohérence**. Si le premier point se rapprochait de nos premiers travaux, mentionnés précédemment, l'ensemble du projet référait à de nombreux paradigmes (cf figure 0.1). Nous allons les évoquer afin d'expliciter le cadre de notre travail, de préciser ceux que nous avons écartés et ceux sur lesquels nous focaliserons notre exposé.



Les types de problèmes représentés ne sont pas indépendants. Il existe ainsi des relations entre non-monotonie et structure des textes, formalisme AVAG et anaphores, connaissances du domaine,... Nous ne les avons pas matérialisés pour ne pas surcharger la figure.

Fig 0.1: Problématique de ACTES

0.2. Spécifications en langage naturel

La phase de définition dans la vie d'un logiciel est de toute première importance [BRACO 88]. Parmi les nombreux langages de spécifications existants, certains ont pour objectif de permettre à l'expert de pouvoir s'exprimer le plus naturellement possible.

[CHEN 82] présente plusieurs de ces langages semi-formels dont certains sont utilisés dans l'aéronautique pour la spécification de problèmes temps-réel. Mais la distance est encore grande entre le mode d'expression de l'utilisateur et le langage offert. De plus, paradoxalement (par rapport à l'approche semi-formelle), les possibilités de mise à jour et de vérifications sont peu mises en avant.

Balzer [BALZ 85] a choisi de développer des langages de spécifications proches du langage naturel pour la programmation. Le bilan de ses quinze ans de travaux montre les difficultés qu'il reste à résoudre pour franchir les étapes allant successivement de spécifications informelles, aux spécifications de haut niveau, bas niveau, et finalement à la compilation automatique.

Alors que les spécifications semi-formelles sont loin d'être maîtrisées, il pourrait sembler prématuré de s'attaquer à celles écrites en langage naturel, sachant qu'à tous ces problèmes de vérifications et de contrôles s'ajoutent ceux liés au traitement du langage naturel et que dans ce dernier domaine les techniques générales font encore défaut. Mais les faits sont là : malgré la floraison d'outils en tous genres d'aide à la spécification et de langages plus ou moins formels, l'essentiel des spécifications est aujourd'hui écrit en langue naturelle et le restera longtemps encore, de l'avis même des experts. Pour limiter au maximum le côté informel de la chose, les experts définissent souvent (comme dans ACTES) le cadre général de l'application avec des méthodes de spécification (SADT, SA, DLAO [CHEN 82]) et n'utilisent qu'un langage naturel contraint pour les descriptions de plus bas niveau. La question est donc posée pour savoir comment traiter informatiquement les textes **existants** écrits directement en langage naturel[2].

Les recherches dans ce domaine sont très récentes et peu nombreuses, citons, par exemple, [GRAN 87], [BIEB 87] et [SELI 85]. Pour le premier, l'analyse se fait phrase à phrase. Le texte n'est pas traité comme un ensemble. Quant au deuxième, il s'agit d'une bonne pré-étude du problème du traitement des spécifications aéronautiques.

---

[2] Entre l'élaboration de langages de spécification semi-formels et le traitement de spécifications écrites en langage naturel (presque) libre, tous les intermédiaires sont sujets d'intérêt. [SAEKI 87] défend, par exemple, l'idée de spécifications utilisant certaines formulations en langage naturel dans un cadre rigide d'énoncés définis en langage semi-formel.

0.3. Actes est-il un projet d'interface pour un système expert ?

Actes n'est pas à proprement parler un projet d'interface pour un système expert, même si les règles que le système complet devrait produire sont destinées à un système expert.

Il ne s'agit pas d'offrir une interface à un expert lui permettant directement de spécifier ses règles et de les valider en utilisant un pseudo langage naturel. Nous ne sommes pas confrontés au problème que rencontrent les cogniticiens lors de l'extraction des connaissances auprès d'un expert, ni au problème de la construction d'un nouveau langage de spécification de haut niveau.

Le langage de spécification est tout trouvé: il s'agit du langage naturel employé dans les textes que nous avons à traiter. Les connaissances des experts y sont inscrites En particulier, la structuration des textes est une source d'informations précieuse pour guider l'extraction des règles. ACTES doit la prendre en compte.

De plus, à la différence du système expert qui doit valider globalement l'ensemble des spécifications, l'architecture que nous proposons peut permettre d'envisager des validations locales portant sur un nombre limité de textes.

0.4. Actes est un projet de compréhension de textes

Actes est un projet de compréhension de textes[3] dont l'architecture cherche à intégrer syntaxe, sémantique, pragmatique, structuration du texte, gestion du discours. Ce pré-requis n'a rien d'original. Depuis quatre à cinq ans des équipes de recherche développent de tels systèmes. Leurs résultats, le nombre de phénomènes linguistiques qu'ils savent prendre en compte dépassent ceux de notre projet, beaucoup plus limité en durée et moyens humains. L'originalité de notre démarche, par rapport à ces systèmes, se situe à deux niveaux, celui des outils employés et celui des objectifs du traitement, à savoir l'extraction de règles.

[3] Une alternative à la compréhension des textes tels qu'ils ont été écrits, serait un système d'aide à la rédaction de règles dans un langage naturel contraint (cf approche guidée par menus dans les systèmes d'interfaces en langage naturel pour les bases de données).

Nous avons tenté d'intégrer nos premiers outils de développement de grammaires utilisant l'unification et la théorie du discours dans un système de compréhension de textes. L'intérêt était de confronter les prétentions généralistes de ces formalismes linguistiques et sémantiques à des problèmes bien concrets liés à un corpus défini et cohérent. La confrontation nous a conduits à modifier ces outils, les étendre, voire les "tordre" pour qu'ils puissent prendre en compte certains phénomènes non prévus initialement. En retour, cela nous permet de discuter des limitations des théories dont ils s'inspiraient et du type d'architecture appropriée à la collaboration de plusieurs théories partielles. Cette boîte à outils intégrée dans un système de compréhension de textes est un exemple de "poste de travail linguistique". Ce terme récent en linguistique informatique renvoie aux environnements de développement de grammaires mettant à la disposition des concepteurs des formalismes pour écrire des grammaires et des modules pour tester et utiliser ces grammaires.

Quant à l'objectif d'extraction de règles, il diffère sensiblement de celui de la plupart des autres systèmes de compréhension de textes qui , eux, s'attachent généralement à extraire des connaissances pour constituer des résumés ou alimenter des bases de connaissances. Certains articles ([HAHN 87] par exemple) attirent l'attention sur ce problème mais nous ne connaissons pas de travaux précis s'y rapportant. L'extraction de règles à partir de textes intéresse, bien sûr, les systèmes de traitement de spécifications et les systèmes experts. Mais nous pensons que le problème concerne également les systèmes de compréhension du langage naturel. L'énoncé en langage naturel des règles se fait par touches successives. Chaque affinement dans la description est susceptible d'apporter des informations contradictoires avec le début de l'énoncé. Bel exemple de non-monotonie, tenant aussi bien à la nature du problème technique qu'à la façon de l'énoncer. Ces ajouts successifs posent aussi un autre problème de traitement du langage, celui de la référence et des anaphores puisqu'il faut être capable de déterminer à quels éléments antérieurs du discours, chaque nouvelle description fait allusion.

Nos travaux sur les deux points, outils généraux de compréhension de textes et extraction de règles, n'en sont pas au même stade d'avancement. Le prototype effectivement réalisé correspond à la première phase du projet ACTES : construction d'un poste de travail linguistique et mise en oeuvre sur un sous-ensemble de textes du domaine en vue de produire les représentations sémantiques intermédiaires des textes analysés. Concernant l'extraction de règles à partir des représentations sémantiques (phase 2 du projet) nous n'apportons que des suggestions sur papier. Seule la résolution des anaphores a été implantée avec une architecture originale par rapport aux autres systèmes de compréhension de textes.

La première partie de cette thèse ne concernera donc que la présentation générale des points en rapport avec le poste de travail linguistique: grammaires d'unification, théorie du discours, anaphores. Nous limiterons principalement la comparaison de nos travaux à ceux des chercheurs travaillant sur ces mêmes points. Le problème de l'extraction des règles sera seulement abordé à la fin de cet ouvrage.

0.5. Hypothèses de travail

Voici la liste de nos hypothèses de travail. Elle servira à justifier le choix de notre approche conciliant grammaires d'unification et théorie du discours. Elle apportera aussi des éléments de comparaison lors de la présentation des grammaires d'unification et des systèmes intégrant des paradigmes voisins du nôtre. Ce ne sont que des indications données au lecteur pour l'aider à lire le rapport de nos travaux. Les hypothèses de travail, en dépit de leur nom, sont souvent extraites à la fin du travail et présentées pour justifier partiellement les a priori de leurs auteurs.

* Même si les textes à traiter appartiennent à un domaine restreint, notre approche doit être suffisamment générale pour pouvoir s'appliquer à la compréhension de textes d'autres domaines.
* Pas de grammaire ad hoc.(pas de grammaire dite *sémantique* ni à base de mots-clés). Les informations linguistiques contenues dans les textes devront être prise en compte
* Le formalisme linguistique devra être le plus indépendant possible d'une théorie grammaticale, ou regrouper les traits communs à plusieurs théories.
* Un seul formalisme pour décrire les informations syntaxiques et sémantiques.
* La représentation sémantique choisie devra rendre compte de certaines caractéristiques propres au texte: relations inter-phrases, structuration du texte, cohésion du texte et phénomènes de références.
* Le formalisme devra offrir des possibilités de représentation des connaissances du monde et/ou se connecter avec un langage de représentation des connaissances.
* Au vu des divers cas d'anaphores rencontrés dans les textes du domaine, une architecture spécifique au module de résolution d'anaphores devra être étudiée en tenant compte particulièrement des problèmes de contrôles de stratégies multiples de résolution, de la place et de l'importance de la sémantique et des connaissances du domaine dans la recherche des antécédents d'un terme anaphorique.
* Le formalisme devra se traduire aisément dans un langage informatique de haut niveau comme Prolog, par exemple.
* L'utilisation d'outils linguistiques généraux devra tendre à libérer la personne qui a la charge de développer une grammaire des contraintes liées à la traduction de cette grammaire dans le langage informatique choisi.
* Les problèmes de stratégies d'analyse devront être distingués des problèmes d'expression des connaissances linguistiques et sémantiques dans une grammaire afin de pouvoir faire varier de façon autonome chacun des paramètres. Le formalisme linguistique devra donc être le plus indépendant possible de toute stratégie particulière d'analyse.

En ce qui concerne la partie extraction de règles et vérification de cohérence, voici les points les plus importants qui nous ont guidés:

* Recherche d'une structure intermédiaire de règle permettant de traduire le contenu d'un texte en suivant un processus séquentiel (correspondant à l'analyse phrase à phrase du texte dans l'ordre de l'énoncé) et en limitant les opérations de réécriture des parties déjà analysées.
* Mise-à-jour possible de cette traduction correspondant à l'énoncé de nouvelles phrases pouvant contredire certains points déjà traités.
* Possibilités de déduction, sur ces structures intermédiaires, afin de pouvoir répondre à des questions de l'expert, du genre: "si je suis dans tel état, que se passe-t-il ?".

0.6. Plan de la thèse

La première partie de la thèse fait un tour d'horizon critique des principales approches du traitement informatique du langage naturel, des grammaires d'unification, de la DRT, de la résolution des anaphores, des systèmes se présentant comme des postes de travail linguistique et des systèmes appliquant les grammaires d'unification et la DRT à la compréhension de textes. Le seconde partie présente nos travaux dans le projet ACTES.

Le **premier chapitre** évoque les différents domaines de recherche dans le traitement du langage naturel. ACTES n'aborde que certains de ces points. Nous faisons ensuite un survol rapide des principales théories du traitement du langage naturel. Chacune apporte un éclairage différent sur ce que sont les processus et les connaissances mis en jeu dans la compréhension d'un énoncé. Ces théories peuvent paraître contradictoires ou complémentaires suivant le point de vue adopté. Aucune n'est complète, au sens où les domaines présentés auparavant et les phénomènes du langage mis en évidence ne peuvent être tous pris en compte dans un cadre unique.

Nous montrons, au contraire, que les systèmes les plus généraux sont ceux qui savent tirer parti de théories diverses en les intégrant dans une même architecture. Nous concluons sur ce que nous pensons être les axes appelés à connaître un développement important dans un proche avenir. Notre travail reprend certains de ces axes.

Le **second chapitre** introduit les grammaires d'unification. Nous avons délaissé le traditionnel survol de plusieurs grammaires de ce type. Il est difficile et frustrant de donner en quelques pages une idée de l'intérêt de telle ou telle grammaire, sans parler de la profusion de notations qu'une telle présentation impliquerait. Nous nous attachons, au contraire, à décrire les concepts "unificateurs" de ces grammaires: les structures de données utilisées pour encoder les données linguistiques et les opérations manipulant ces structures et traduisant les contraintes de la langue. Nous insistons sur la représentation de connaissances (notamment celle du domaine d'un texte) au moyen de types.

Cette présentation, un peu informatique, des grammaires d'unification nous permet de faire le parallèle avec le langage de programmation Prolog que nous utilisons, et plus généralement de notions familières en programmation logique. Les grammaires d'unification (du moins certaines d'entre elles) se présentant souvent comme des formalismes linguistiques déclaratifs, à la sémantique claire et facile à implanter, il nous paraît intéressant de montrer ce qu'elles ont de commun avec certains langages de l'informatique et ce qui les distingue de ces langages, en laissant de côté certaines variantes terminologiques qui n'apportent rien de neuf.

Le **troisième chapitre** parle de sémantique, sémantique d'un énoncé, représentation sémantique d'un texte, et de résolution d'anaphores. Caractériser la sémantique du langage est une chose très complexe. Il n'est pas sûr que vouloir trouver une procédure de décision pour savoir si un énoncé a ou non un sens soit la bonne façon de poser les problèmes. Particulièrement les approches dites sémantiques formelles, qui se fixent l'objectif de caractériser le sens *en soi* d'un énoncé ont souvent des présupposés très forts sur la nature de la langue et, pour certaines, une vision réductrice du problème. Nous détaillons la Théorie de la Représentation du Discours que nous utilisons dans ACTES parce qu'elle est une des rares approches permettant de construire une représentation sémantique intermédiaire reliant plusieurs phrases d'un même texte et qu'elle est d'implantation aisée.

La traduction des relations entre plusieurs phrases d'un énoncé passe par la prise en compte du contexte et notamment par la résolution des problèmes de références et d'anaphores. Un survol général rend compte de la diversité des connaissances et des processus mis en jeu dans la résolution d'anaphores, nombreuses dans les textes de ACTES. Au vu de cette présentation, nous revenons sur les affirmations de la DRT en indiquant quelques limites que nous tentons de pallier dans notre système.

La première partie s'achève avec le **quatrième chapitre**. Nous y présentons des systèmes de traitement du langage naturel qui se situent dans une approche voisine de la nôtre, c'est-à-dire des systèmes qui ont été construits à partir d'hypothèses proches de celles énoncées précédemment. Nous évoquons d'abord quelques grammaires d'unification qui ont choisi la Théorie de la Représentation du Discours comme représentation sémantique ; puis des environnements de développement de grammaires d'unification ; et enfin quelques systèmes de compréhension de textes construits à partir de grammaires d'unification et/ou utilisant la Théorie de la Représentation du Discours.

Ce tour d'horizon original des systèmes implantant grammaires d'unification et DRT a pour but de savoir si des traits les distinguent d'autres approches plus conventionnelles en traitement du langage et de dégager quelques critères de comparaisons permettant de mieux situer ensuite notre travail. A cette occasion nous évoquons les principales approches en compréhension de textes.

Le **chapitre cinq** introduit la deuxième partie Il décrit l'architecture générale du système ACTES. Puis il indique suivant quels critères, après une étude linguistique générale de tous les textes du domaine, nous avons sélectionné un sous-ensemble pour tester le prototype. Un exemple de texte est donné en illustration. Un scénario d'utilisation éclaire le cheminement procédural que doit suivre l'utilisateur de notre système et explicite ainsi les liens logiques entre les différents modules de ACTES.

Le **sixième chapitre** est consacré au formalisme de développement de grammaires AVAG. Ce formalisme permet de construire une représentation sémantique intermédiaire des textes. Il s'inspire des grammaires d'unification et de la Théorie de la Représentation du Discours. L'approche mise en oeuvre ici présente l'avantage d'être unifiée: les connaissances syntaxiques et sémantiques sont représentées dans les mêmes structures de traits et manipulées par l'opération d'unification de Prolog. Toutefois, Prolog ne prenant pas en compte les mécanismes d'héritage, une opération d'unification étendue gérant les contraintes hiérarchiques de types entre éléments du formalisme a été définie. On peut ainsi récupérer les connaissances du domaine décrites dans la partie terminologique du langage hybride de représentation des connaissances BACK couplé à notre système.

La présentation de BACK nous offre l'occasion de parler des recherches en représentation des connaissances. La représentation des connaissances du domaine est un point important des systèmes de traitement du langage naturel mais est ignoré dans certaines approches sémantiques. Nous rappelons les travaux de ces chercheurs en Intelligence Artificielle qui savent à la fois discuter sémantique des langages et efficacité informatique. Nous discutons également la question de l'utilisation de langages généraux de représentation des connaissances ou du développement d'un formalisme ad hoc aux besoins du système.

Le **septième chapitre** se rapporte à l'environnement du poste de travail. Cet environnement aide l'utilisateur à tester les grammaires écrites dans le formalisme AVAG et à les utiliser pour l'analyse de textes. Il prend en charge le découpage des textes en 'tokens' élémentaires et offre une interface conviviale avec le langage de représentation des connaissances.

Nous discutons de façon critique des solutions originales que nous avons développées ou de celles que nous avons reprises à d'autres systèmes. Cette discussion concerne les aspects structuration des textes, diagnostics d'erreurs et stratégies d'analyse.

Puis nous développons particulièrement l'architecture du module de résolution d'anaphores de type "tableau noir" en la comparant à celles d'autres systèmes de TLN. Elle nous permet, en particulier, de pallier certaines insuffisances de la Théorie de la Représentation du Discours en intégrant ses contraintes en matière d'anaphores dans un module du tableau noir dont les conclusions peuvent être remises en cause par d'autres composants de l'architecture.

Le **dernier chapitre**, enfin, parle du problème de l'extraction des règles à partir des textes retenus pour le prototype. Le système expert simulant les alarmes est à base de règles de production. Plutôt que de générer directement de telles règles, nous proposons d'essayer de produire automatiquement des règles de défaut à partir de la représentation sémantique intermédiaire.

Ce choix est motivé par la correspondance entre la représentation du processus d'extraction des règles dans un texte et la non-monotonie qui se formalise bien dans la logique des défauts. De façon plus générale, le traitement automatique de textes en langage naturel s'assimile bien à un processus non-monotone. Enfin, et surtout, nous disposons d'un cadre pour envisager la vérification interactive de cohérence des textes.

Après un bref rappel sur la logique des défauts, nous illustrons, à partir du texte donné en exemple, les avantages par rapport à une formalisation en logique du premier ordre. Dans une deuxième partie, nous évoquons les tâches à accomplir pour étendre ACTES en vue de réaliser un système de vérification de cohérence basé sur les défauts.

1. INTRODUCTION AU DOMAINE DU TRAITEMENT DU LANGAGE NATUREL

"Our field exists because of one natural phenomen, human language, and one technology, the computer."

"Why don't a group of us take the best parser, the best semantic interpreter, the best generator, the best inference system, etc. , and tie them together ? Then let's pick a domain of discourse and make them work for more than a few sentences. Let's beat on them until they work as much of language as they appear capable. ... Let's make the system as fast, as robust, as portable, as maintable, etc., as we possibly can."

SONDHEIMER N.M., FORUM : "WHY HAS THEORETICAL NATURAL LANGUAGE PROCESSING MADE SO LITTLE PROGRESS ?", TINLAP 3, 1987.

1.1. PROBLEMATIQUE DU TRAITEMENT DU LANGAGE NATUREL

Avant d'introduire les différentes théories sur le traitement informatique du langage naturel (désormais abrégé en TLN), nous allons présenter (de façon quelque peu schématique pour la simplicité de l'exposé) les grands domaines de recherches actuelles en TLN:

- les modèles syntaxiques
- l'interprétation sémantique
- l'interprétation du discours
- action et intention
- la génération
- la construction de systèmes englobant les aspects précédents
- les approches connexionnistes.

1.1.1. Les modèles syntaxiques

Le plus ancien et le mieux étudié des domaines du TLN est celui relatif à la syntaxe des langages naturels : elle décrit la façon dont les mots d'une phrase se regroupent en constituants et les constituants en constituants plus larges, ainsi que les relations entre ses constituants.

La recherche dans ce domaine a longtemps concerné deux notions différentes: les grammaires et les algorithmes d'analyse[1] ([WINO 83] [SELL 85 b]).

Une grammaire est une description finie d'un langage potentiellement infini qui capture ses régularités. Elle peut être utilisée pour générer les phrases d'une langue ou déterminer si une chaîne donnée appartient à un langage et , dans ce cas, déterminer sa structure Les algorithmes d'analyse spécifient comment appliquer une grammaire à une suite de mots de façon à produire une représentation structurée, l'arbre d'analyse (ou arbre syntaxique). Un analyseur regroupe une grammaire et un algorithme d'analyse (nous n'abordons pas, pour l'instant le problème de l'introduction de la sémantique ou de la pragmatique en cours d'analyse).

[1] Nous verrons que dans les nouvelles approches, notamment celles des grammaires d'unification, on a tendance à séparer le développement de formalismes grammaticaux des algorithmes d'analyse.

Le travail sur les modèles syntaxiques pour les systèmes de TLN est bâti sur les résultats de la linguistique théorique (les formalismes utilisés pour caractériser les chaînes grammaticales du LN) et de l'informatique (résultats sur la théorie des langages formels, sur la compilation). Tous deux sont concernés par les problèmes de puissance d'expression. Cependant, tandis que la linguistique théorique recherche les universaux du langage, l'apprentissage formel (élaboration d'une grammaire à partir de données linguistiques), l'adéquation psychologique et la simplicité, les théories de la compilation s'intéressent plus au problème de contrôle et d'efficacité. L'analyse automatique du LN est concerné en plus par les problèmes de couverture linguistique[2].

1.1.2. Interprétation sémantique

En logique, la sémantique est l'opération de mise en correspondance d'un langage avec le monde ou un modèle correspondant à un monde, réel ou imaginaire. En TLN, cette correspondance n'est pas faite directement: il faut d'abord représenter le sens d'un énoncé avant d'interpréter cette représentation sur le monde ou un modèle. L'interprétation sémantique en TLN diffère donc de l'interprétation sémantique de la logique parce qu'elle s'intéresse à la production, à partir d'un énoncé, d'une **représentation intermédiaire du sens**.

Si cette représentation du sens d'un énoncé est complète, elle prend en compte le domaine, le contexte, les actions et les tâches. Le **domaine** est la description partielle du monde (réel ou imaginaire) connu du système: quels sont les objets, les relations entre ces objets (connaissances terminologiques), quels sont les faits considérés comme vrais décrivant les propriétés de certains objets (connaissances assertionnelles). Le **contexte** d'un énoncé dépend des énoncés précédents du discours, du lieu et du moment où est produit l'énoncé, des croyances, intentions, désirs des participants. Une **tâche** est le service que le système est susceptible d'offrir à l'utilisateur (recherche d'informations, aide à la prise de décisions,...), et **l'action** est ce que l'utilisateur désire accomplir.

La représentation complète d'un énoncé n'intègre donc pas seulement des connaissances locales liées au lexique ou à la grammaire, mais également des information sur le discours, des connaissances générales sur le monde et sur les intentions des locuteurs. Si le processus de construction de la représentation sémantique ne fait intervenir que des connaissances locales, le résultat peut être ambigu ou indéterminé. L'**ambiguité** peut être structurelle (plusieurs constructions possibles de la représentation à partir de ses constituants - cf problème de portée des quantificateurs de la langue -) ou lexicale (sens multiples d'un mot) L'**indétermination**, elle, est relative aux connaissances implicites d'un énoncé.

---

[2] ensemble des types de construction reconnus par la grammaire tels les affirmatives, interrogatives, relatives, ...

Notons que pratiquement tous les modèles (non logiques) de représentation des connaissances utilisés pour décrire domaine, contexte, action, tâche et forme de la représentation sémantique intermédiaire sont issus des travaux d'Intelligence Artificielle (désormais notée IA) [BRAC 85a]. Ce sont aussi souvent les chercheurs en IA qui ont, les premiers, appliqué les modèles de la logique mathématique au traitement informatique du langage. D'autre part, certains linguistes et philosophes du langage ont été inspirés directement par les travaux des mathématiciens [CAWL 81]. Leur approche est souvent qualifiée de **sémantique formelle** et vise à la mise en correspondance d'une représentation du sens d'un énoncé avec le monde ou un modèle du monde. Cette façon d'opposer ces deux approches à la construction du sens d'un énoncé est quelque peu schématique. Ainsi, dans notre système, nous utiliserons la Théorie de la Représentation du Discours (qui peut être qualifiée d'approche sémantique formelle - cf partie correspondante -), principalement comme une représentation sémantique intermédiaire.

1.1.3. Interprétation du discours

Si les premières approches en TLN se limitaient à un traitement de phrases isolées, aujourd'hui, une partie notable des systèmes abordent le traitement de textes ou discours. Nous entendons le mot texte comme un ensemble de phrases (généralement sur support écrit) se rapportant à un sujet donné. Un discours sera considéré comme une extension de la notion de texte: plusieurs participants peuvent être impliqués (humains ou homme-machine), le support langagier peut être vocal ou écrit (ce dernier point nous intéressera principalement).

L'interprétation d'un discours s'attache à la description et la prise en compte du contexte d'un énoncé. Les recherches sur ce domaine ont montré l'importance prépondérante des facteurs non linguistiques: suivi du discours, phénomènes de cohésion et cohérence d'un discours. Dans l'étude des relations entre les phrases composant un discours, les phénomènes de **référence** occupent la première place.

1.1.4. Action et Intention

La présentation de la gestion du discours présentée dans le paragraphe précédent s'intéressait à la description du contexte en ne prenant en compte que les aspects liés aux phénomènes de référence. Dans cette partie on cherche à prendre en compte les croyances et intentions des différents partenaires d'un discours et à les relier avec les actions qu'ils désirent entreprendre (en fait les différents aspects du contexte sont interdépendants).

Les travaux sur les aspects *action et intention* sont conduits par les philosophes du langage et les chercheurs en IA. Les premiers défendent l'idée que le langage est utilisé en vue d'accomplir des actions. Pour arriver à comprendre le déroulement d'un discours, identifier les buts poursuivis par chacun des participants, le système doit tenir compte des croyances, expliciter les intentions des différents partenaires. Ils ont ainsi mis en évidence les **règles** utilisées tacitement par chaque participant désirant **communiquer** avec ses partenaires. L'Intelligence Artificielle, elle, identifie conduite d'un discours et tâches de **planification**. Le cadre général du discours indique le plan d'ensemble. Chaque partie ou paragraphe décrivant une digression ou un changement de sujet est traitée comme la construction de sous-plans ou le changement du plan initial. Pour ce faire, il a fallu adapter les techniques de planification utilisées, au départ, pour la conduite des robots.

1.1.5. Génération

Initialement la génération n'a été perçue par les chercheurs que comme un complément à l'analyse. Un système générait des phrases pour montrer qu'il avait correctement analysé celles données en entrée. Il s'agissait alors de convertir une représentation sémantique intermédiaire en phrase isolée du contexte du discours.

Lorsque l'on a voulu construire des systèmes qui prenait en compte un discours et la pragmatique associée et qui était capable d'analyser comme de générer un ensemble de phrases, la problématique de la génération s'est très vite complexifiée et décomposée en trois sous-problèmes ([RNLP 86], [ZOCK 88]): détermination du contexte, planification du texte, construction d'une suite de phrases. Les deux premiers représentent le coté *stratégie* de la génération, le troisième, le côté *tactique*.

La **détermination du contexte** s'intéresse au problème d'identifier les principaux constituants devant figurer dans le texte à générer. Dans un dialogue homme-machine, par exemple, le système opère des choix en fonction de ce qu'il sait du contenu de la réponse au problème posé initialement et de ce qu'il sait des connaissances ou intentions initiales de l'utilisateur.

La **planification du texte** à générer s'occupe de l'agencement des différents composant identifiés précédemment de façon à construire un exposé cohérent et respectant un certain style rhétorique.

Enfin la troisième partie du problème consiste, à partir du plan détaillé élaboré précédemment, à sélectionner **les mots pour le dire** et construire les structures syntaxiques correspondantes.

1.1.6. Systèmes de TLN

Dans un système de TLN, il s'agit de mettre ensemble et faire fonctionner les différents éléments identifiés dans les paragraphes précédents[3]. Les systèmes de TLN peuvent être rangés en deux grandes classes: systèmes interactifs où le langage naturel est l'un des modes d'interaction homme-machine (au côté du graphique, du son, des systèmes de désignation comme la souris,...). et système de compréhension de textes (ou traitement automatique de textes[4]) visant à extraire des connaissances d'un texte pour mettre à jour une base de données, traiter des messages, construire des résumés, les traduire d'une langue dans une autre,... Cette dernière classe nous intéressera particulièrement dans notre travail. Certains systèmes sophistiqués intègrent bien sûr les deux classes de problèmes.

Lorsque l'on s'intéresse au traitement du langage naturel en ayant en vue la construction d'un système informatique, certains résultats apparaissent essentiels: modularité, intégration, factorisation, portabilité, habitabilité, extensibilité, vitesse, adéquation cognitive. Précisons quelques-uns de ces points.

La modularité concerne la séparation des informations dépendantes du domaine de celles qui ne le sont pas et l'indépendance entre les différents modules de traitement du système. C'est l'architecture du système qui indique le degré d'intégration de ses composants: analyse séquentielle ou parallèle, analyse partielle avec mise en commun des résultats ou élaboration par étapes successives,... Le degré de factorisation dépend du partage des responsabilités entre modules du système pour traiter tels ou tels phénomènes linguistiques. L'habitabilité dépend de la couverture du système sur un domaine donné et ses capacités à réagir face à des énoncés hors-domaine.

1.1.7. Approches connexionnistes

Depuis un an ou deux, l'approche "réseaux neuronaux" ou connexionnisme [SELM 89] occupe une place de choix dans les conférences/congrès d'IA et de TLN. Le thème est redevenu à la mode, après plusieurs dizaines d'années de disparition, sans doute, grâce aux récents progrès en technologie électronique: il devient possible de concevoir des puces dont l'architecture simule certains aspects du fonctionnement des neurones humains.

---

[3] aujourd'hui ,les applications en TLN développées dans les laboratoires de recherche universitaires ou industriels qui méritent le nom de systèmes de TLN intègrent bien ces différents aspects. Ils sont construits par des équipes dont chaque membre est spécialisé dans un aspect particulier.

[4] Nous n'utiliserons pas le terme plus simple de 'traitement de textes' car il peut prêter à confusion avec les logiciels dont celui utilisé présentement pour écrire ce texte. En anglais deux termes distincts sont employés *text processing systems* (compréhension de textes) et *word processing* (traitement de textes).

En calculant de façon stochastique et non plus déterministe, certains de ces réseaux semblent capables de trouver de bonnes solutions approchées à des problèmes réputés difficiles (reconnaissance de formes, en particulier de caractères écrits). Quelles peuvent être les applications au TLN ? Waltz [WALT 87] défend les idées suivantes :

- la capacité des réseaux à calculer la plus proche coïncidence plutôt que l'unification ou l'exacte coïncidence peut être utilisée pour résoudre certains cas d'ambiguïtés lexicales.

- l'architecture se prête bien aux problèmes d'apprentissage (mots nouveaux ?), mais une grande difficulté réside dans la définition des connaissances minimales à implanter dans le réseau.

- La modularité et la tolérance aux fautes peut permettre la reconnaissance d'expressions mal formées difficile à traiter avec les analyseurs habituels, en accédant à des "îlots" de connaissances partielles syntaxiques, sémantiques, pragmatiques et en recoupant ses connaissances.

Aujourd'hui aucune expérience sur une échelle comparable aux autres approches traditionnelles en TLN n'a encore pu être réalisée. Il est donc difficile de se faire une idée précise des potentialités de cette approche. L'absence de manipulations symboliques, remplacée par des messages très simples d'inhibition/activation peut laisser penser que l'approche connexionniste sera difficile à utiliser en représentation des connaissances où le savoir s'exprime en termes de concepts, et au contraire, plus facile dans les problèmes faisant appel habituellement à des systèmes de pondération numériques (approche statistique en TLN, reconnaissance de formes). Cependant Brachman [BRAC 88] a essayé de trouver un lien entre les approches 'représentation des connaissances' et connexionniste et l'appliquer à la recherche d'information dans un texte, en décrivant les concepts d'un domaine au moyen d'ensemble de micro-traits. A suivre ...

1.1.8. Remarques

Les ouvrages de référence généraux en traitement du langage naturel sont le livre de Allen, *Natural Language Understanding* [ALLE 87], le volume *Readings in Natural Language Processing* [RNLP 86] recueillant les meilleurs articles sur les 6 domaines évoqués et les deux tomes du livre de Sabah, *L'intelligence artificielle et le langage* ([SABA 88], [SABA 89]) Un bon article général peut également être trouvé dans [COUL 86]. La publication périodique la plus connue en linguistique informatique est la revue *Computational Linguitics* souvent référencée dans ce travail.

Des aspects important se rapportant au traitement du langage naturel n'ont même pas été évoqués. Voici quelques références se rapportant dialogue vocal [PIER 87], aux approches statistiques [SALT 88], à la traduction automatique [CL 85].

1.2. APPROCHES THEORIQUES ACTUELLES EN TRAITEMENT DU LANGAGE

Nous allons évoquer succinctement les principales théories du traitement du langage naturel, donnant lieu aujourd'hui à des applications informatiques. Afin d'éviter toute équivoque, précisons les termes de la phrase précédente.

Il ne s'agit pas de faire un survol des différents courants de la linguistique contemporaine, mais de ceux que l'on classe généralement sous l'appellation "linguistique informatique" (*computational linguistics*). L'adjectif "informatique" a pour nous beaucoup d'importance puisque nous examinerons les approches qui sont totalement ou partiellement reprises dans des systèmes informatiques de traitement du langage naturel. Nous ne nous intéressons pas ici aux implantations qui seraient des simulations de théories particulières, mais à celles qui apportent des éléments de réponses pratiques sur plusieurs des points évoqués dans la partie précédente (syntaxe, sémantique, discours,...).

Le mot "théorie" est à prendre au sens large, référent à un ensemble d'idées, de concepts abstraits, plus ou moins organisés appliqué au TLN. Notre survol ne saurait être complet, ni présenter en quelques lignes chacune des théories. Il se contentera d'en indiquer les traits caractéritiques et les références les plus significatives au lecteur désireux d'approfondir ces sujets. Les théories citées ne sont pas nécessairement les plus récentes, mais celles qui, aujourd'hui, inspirent les systèmes de TLN (dont ceux qui ont pour objectif la compréhension de textes) que nous estimons les plus complets ou les plus prometteurs.

Les théories chomskiennes sont également mentionnées parce qu'elles ont influencé pendant des années tout un courant de la linguistique informatique et qu'une partie des concepteurs des grammaires d'unification (cf chapitre 2) situent leurs travaux par rapport aux limites des théories grammaticales élaborées par Chomsky avant les années 80.

Une conclusion critique nous permettra d'introduire le cadre de notre travail.

1.2.1. Analyse distributionnelle et sous-langages

Harris est l'un des premiers à avoir mener des travaux rigoureux et détaillés sur la syntaxe des langues [HARR 51]. Il pensait que toute théorie de la langue devait s'appuyer sur une étude exhaustive (distributionnelle) des phénomènes qu'on y observe.

Il développe en 1968 la théorie des grammaires en chaînes [HARR 68] que Sager transformera en vue d'implanter un sous-ensemble significatif de l'anglais [SAGE 81] (cf [SALK 73] pour une mise en oeuvre de la grammaire en chaîne du français).

Les éléments de la grammaire sont des structures syntaxiques qui sont nommées des chaînes. Les chaînes sont constituées d'une suite de catégories grammaticales définies de manière axiomatique par l'analyse distributionnelle: nom, verbe, adjectif, adverbe. Un ensemble de chaînes centrales élémentaires constituent le noyau de la grammaire: ce sont les formes globales des phrases simples du type sujet-verbe-objet.

D'autre part des chaînes d'ajout sont définies. Elles correspondent aux modificateurs de la grammaire traditionnelle et peuvent s'insérer dans une chaîne centrale ou non. Les phrases de la langue sont alors des chaînes dérivées des chaînes centrales par insertion de chaînes d'ajout[5].

Très tôt (1969 ?) les linguistes de ce courant ont essayé de construire des systèmes de traitement automatique de textes de domaines spécialisés ([KITT 82], [GRIS 86]): médecine, chimie, météo, documents de maintenance techniques,... Les textes produits par les experts de ces domaines forment un sous-langage. Le problème es de séparer les régularités propres au domaine (en liaison directe avec les connaissances du monde) de celles du langage en général. Avec la méthodologie d'analyse distributionnelle, partant des phrases de la langue, ils définissent la façon dont les réductions opèrent pour passer de la phrase aux chaînes centrales. Des sous-classes sémantiques et des propositions types traduisent les contraintes rigides sur la façon dont les mots peuvent être assemblés dans une phrase, et réduisent ainsi fortement les ambiguïtés syntaxiques. Ces sous-classes et ses propositions se traduisent alors aisément en termes de concepts dans un réseau sémantique et de prédicats logiques ou liens entre concepts.

---

[5] insertion est un peu réducteur: les mouvements de chaînes peuvent être des transposition, modifications (morphologiques), enlèvement (chaînes optionnelles).

La grammaire en chaîne a été fortement décriée par divers courants en linguistique informatique (multiplicité des règles, faible efficacité des premiers analyseurs, sous-estimation de la sémantique ...). Or, aujourd'hui l'une des références en compréhension de textes est le système PUNDIT [HIRS 88] développé par des chercheurs de ce courant. Sans détailler trop des points sur lesquels nous reviendrons plus loin, ce système est composé d'une grammaire développé à partir de celle de Sager, couplé avec un réseau sémantique et un systèmes de règles sémantiques pour gérer les connaissances implicites, pourvu d'un système de gestion du discours pour traiter notamment les problèmes anaphoriques.

1.2.2. Grammaires syntagmatiques, génératives, transformationnelles

Impossible de passer sous silence les travaux de Chomsky, même si ses théories n'ont donné lieu à l'implantation d'aucun système de TLN convaincant jusqu'à ce jour. D'abord parce que Chomsky a apporté une caractérisation formelle de la notion de grammaire dont les informaticiens se servent couramment pour construire des langages artificiels: grammaires régulières (type 3), indépendantes du contexte ou hors-contexte (type 2), sensibles au contexte (type 1) et grammaire de type 0. L'autre raison est l'influence considérable de Chomsky sur tout le courant linguistique informatique passé et actuel.

Lorsque, à la fin des années 50, Chomsky a publié ses travaux sur les grammaires formelles et les grammaires transformationnelles [CHOM 57], il voulait répondre à deux types de problèmes : d'une part, la démarche de linguistique structurale de cette époque fournissait une description des langues sans expliciter les régularités fondamentales du langage ; d'autre part, les grammaires syntagmatiques (limitées alors aux grammaires hors-contexte)[6] étaient trop limitées pour décrire certaines constructions syntaxiques.

Chomsky rejette alors le point de vue des structuralistes pour adopter celui des *grammaires universelles* dont le but est d'expliquer l'aspect créateur du langage humain. Le langage va devenir un objet d'étude coupé du contexte et décrit à l'aide de connaissances purement syntaxiques : l'aspect **réalisation** (*performance*) du langage, qui s'intéresse à ce que dit un locuteur et la façon dont il le dit, est écarté au profit de l'étude de l'aspect **compétence** linguistique , caractérisation abstraite de la façon dont un locuteur reconnaît immédiatement comme appartenant à sa langue une phrase donnée, et produit ou comprend une phrase qu'il n'a jamais produite ou entendue auparavant.

---

[6] Les grammaires syntagmatiques (*phrase structure grammars*) (origine ?) traduisent au moyen de règles de réécriture les relations entre syntagmes d'une phrase. Chaque syntagme pouvait être décomposé en sous-syntagmes. Nous identifierons syntagmes et catégories grammaticales. Une catégorie grammaticale correspondra, en première approximation, à un symbole préterminal du langage (catégorie lexicale comme: nom, adjectif, verbe,...) ou sera elle-même composée de catégories (comme le groupe nominal). Les symboles terminaux du langage sont les mots de la langue. Les règles de réécriture hors-contexte ont un et un seul symbole non terminal dans leur partie gauche.

Cette façon de considérer l'aspect créateur du langage (elle caractérise la linguistique **générative**) amène à voir le langage comme un objet mathématique. Une phrase sera jugée grammaticalement correcte, s'il existe une dérivation qui démontre que sa structure est en accord avec un ensemble de règles, comme une preuve démontre la vérité d'une expression mathématique.

Parce que Chomsky apportait des outils formels pour étudier le langage, le milieu linguistique l'a suivi jusqu'au bout de cette démarche réductrice. Le problème devenait uniquement la caractérisation des phrases **correctes** du langage et non des phrases **acceptables**. En écartant l'étude de constructions agrammaticales, mais qui pouvaient être cependant compréhensibles, on s'écartait de l'étude du sens du LN. L'étude du langage naturel se rapprochait de celle des langages artificiels. Ce point de vue subsiste encore aujourd'hui dans une partie du milieu linguistique informatique (cf les objectifs des grammaires récentes comme les grammaires syntagmatiques généralisées). Cela explique sans doute le peu de progrès réalisés sur le traitement des expression mal formées [CL 83], particulièrement par les tenants de l'approche syntaxique.

Pour répondre aux limitations des grammaires hors-contextes, Chomsky a développé la théorie des grammaires transformationnelles (cf [JACO 78] qui est une des meilleures introductions à ce sujet). Sans détailler, disons simplement qu'une grammaire transformationnelle, dans la théorie du début (théorie standard) a deux composants essentiels:

- une grammaire formelle hors-contexte permettant d'engendrer un ensemble de structures abstraites dites structures profondes, correspondants en gros à des phrases simples déclaratives auxquelles on associe un arbre de dérivation.
- un ensemble de règles de transformations, agissant sur les arbres de dérivation et permettent d'engendrer la structure de surface de toutes les formes possibles de phrases.

Un des plus graves défauts de cette première approche est sa puissance d'expression beaucoup trop grande (équivalente à une grammaire de type 0). Virtuellement, une grammaire quelconque couplée avec des règles de transformation bien choisies peut engendrer n'importe quel langage. Chomsky apportera plusieurs modifications importantes à cette théorie, en contraignant les transformations de façon à limiter la puissance du langage et en introduisant une composante sémantique : théorie standard étendue, théorie des traces, et théorie du Gouvernement et du Liage ([CHOM 81] et une bonne introduction dans [SELL 85 b]) sur laquelle travaillent aujourd'hui nombre de linguistes. Comme nous l'avions signalé en introduction, ces travaux n'ont donné lieu qu'à peu d'implantations[7], sans doute parce que Chomsky n'a jamais travaillé avec cette finalité. Ses première théories présentaient trop d'inconvénients techniques et une mauvaise articulation avec la sémantique[8].

1.2.3. Approches sémantiques

1.2.3.1. Primitives sémantiques et organisation de la mémoire

Dans l'approche de Schank et de son équipe à l'université de Yale [SCHA 87] l'accent est mis sur le processus de compréhension, pas sur le langage 'en soi'. Le langage est un moyen de communiquer un contenu, des connaissances qui souvent ne sont pas explicitement formulées. Le problème est donc de s'attacher à répondre aux questions sur la façon dont nous comprenons et représentons les concepts que le langage peut véhiculer, comment nous apprenons de nouveaux concepts, comment cette connaissance est organisée en mémoire.

Ses travaux commencent au début des années 1970 par l'élaboration d'analyseurs guidés par la sémantique, en l'occurrence des représentations à base de cas appelées Dépendances Conceptuelles. L'idée est que certains mots , surtout les verbes, identifient des structures sémantiques ayant des cas composés à partir de primitives sémantiques. Le travail essentiel de l'analyseur consiste à reconnaître ces mots et à utiliser le reste de la phrase pour remplir les valeurs de ces cas. Ce processus de reconnaissance est guidé par des règles d'inférences associées à des schémas de Dépendances Conceptuelles. L'originalité du travail réside dans la définition de ces primitives sémantiques et des opérations de base pour les manipuler. Elles doivent refléter le niveau de pensée qui sous-tend le langage, plutôt que le langage lui-même.

---

[7] à noter toutefois les travaux de Marcus à partir des traces [MARC 78] visant à implémenter un des premiers analyseurs déterministes qui reste aujourd'hui encore une référence.

[8] Sa théorie du liage répond en partie aux insuffisances précédentes en réservant notamment une place de choix à la sémantique et au lexique. D'autre part des implantations de cette théorie sont en cours aujourd'hui.

Pour prendre en compte le contexte et disposer d'un niveau de prédiction qui ne soit plus limité à la simple phrase mais applicable à un texte, Schank développé ensuite une théorie basé sur des scripts et plans [SCHA 77] censés correspondre à des structures de la mémoire. Les scripts (ou scénarios) sont des structures de connaissances statiques décrivant tous les aspects (stéréotypes) d'une situation donnée. Le planificateur reconnait les scénarios invoqués, identifie les étapes mentionnées, remplit les parties implicites dans le texte. Cette théorie permet de construire certains des premiers systèmes de compréhension de petites histoires SAM [CULL 78] et FRUMP en 1977 [DEJO 82].

Cette approche connait alors un grand succès mais présente certains inconvénients, comme la difficulté de définir les conditions de déclenchements d'un scénario, de prendre en compte les situations anormales ou encore de gérer les interactions entre scénarios. Il faut mentionner aussi la grande quantité de connaissances systématiquement appelées lors de la compréhension d'une situation, alors que souvent des connaissances superficielles pourraient suffire[9]. Schank s'intéresse alors à la façon dont les scénarios sont organisés en mémoire et sont appris dynamiquement, en introduisant des schémas organisationnels, les MOPS [SCHA 80], pour connecter ces structures.

Ainsi les tendances des recherches de l'équipe de Yale les conduisent à rejeter l'idée que la compréhension du langage est un processus guidé par les données de mise en correspondance de phrases avec des représentations internes. Pour eux le processus est d'abord guidé par la structure de la mémoire.

Beaucoup de reproches, venant du milieu linguistique comme du milieu IA, ont été adressés à Schank. Ils portaient essentiellement sur le manque de formalisation des structures sémantiques décrites dans ses théories, ainsi que leurs connotations fortement psycho-cognitives (donc "gratuites" aux yeux de certains).

Il n'empêche qu'aujourd'hui certains des systèmes de compréhension de textes, "à l'état de l'art", développés en milieu industriel et s'appliquant à des textes réels, l'ont été par des élèves de Schank et reprennent une partie de son approche. Pour assurer une certaine efficacité ils ont développé des architectures multi-stratégies multi-connaissances (utilisant, par exemple, des méthodes statistiques et des connaissances linguistiques). Citons ainsi le système ATRANS [LYTI 86] de traitement de télex bancaires et SICSOR [RAU 88] sur les annonces financières de prise de participation de sociétés.

---

[9] Kayser et Castaing [CAST 88] abordent le problème du rapport entre le niveau de description des connaissances et le niveau de compréhension en développant l'idée de raisonnement à profondeur variable.

1.2.3.2. La sémantique des préférences de Wilks

Un des travaux les plus célèbres sur l'incorporation de la notion de préférences dans l'interprétation sémantique est celui accompli par Wilks ([WILK 75], [WILK 78], [WILK 85]). Dans son système l'information sémantique est composée d'un ensemble de schémas/gabarits (*templates*) sémantiques définissant les relations sémantiques possibles dans le système et d'une formule sémantique associée à chaque entrée lexicale. L'opération essentielle consiste à faire correspondre ces schémas à une représentation de la phrase de laquelle a été extraite le syntagme principal. Le point important est que des correspondances partielles sont autorisées. Le système produit un groupe de schémas correspondant partiellement chacun à une fraction de la phrase d'entrée. Il sélectionne ensuite la combinaison de schémas qui introduit le plus petit nombre de violations de types.

Wilks a d'abord appliqué ses idées à un système de traduction automatique qui fonctionnait sur un ensemble de phrases d'une longueur de 20 à 30 mots et savait résoudre certains cas de références. La taille et les difficultés du corpus étaient exceptionnelles en 1975. Il a montré de façon convaincante que pour résoudre ces cas de référence, l'approche des grammaires transformationnelles était inadéquate parce qu'incapable de prendre en compte les relations sémantiques essentielles pour opérer des choix entre les mots. Combattant également l'approche uniquement générative/dérivationnelle de ces grammaires il a, par ses méthodes de correspondances partielles, implanté l'un des premiers systèmes capables de comprendre certaines phrases mal formées ou ambiguës. Cependant son système était lent et peu efficace: en particulier la sélection des schémas à partir des mots de la phrase était couteuse car elle ne tenait aucun compte des informations syntaxiques.

Carter [CART 87] a construit un système de résolution de référents contenus dans des textes en utilisant l'approche sémantique de Wilks. Son architecture permet de pallier en partie les défauts de la sémantique des préférences. Il a assemblé un analyseur syntaxique développé par Boguraev, les schémas et règles d'inférences de Wilks et un module de gestion du discours construit à partir des idées de Sidner. La prise en compte de toutes ses informations lui permet d'appliquer une tactique de traitement superficiel (*shallow processing*) en utilisant au mieux les contraintes syntaxico-sémantiques et les contraintes pragmatiques liées à la gestion du discours pour sélectionner les antécédents potentiels d'un référent. Les règles d'inférences de Wilks n'interviennent alors qu'en dernier recours. Ainsi aucune connaissance n'est sous-estimée et les soucis de performance minimisent l'accès à des ressources critiques.

1.2.3.3. Autre approches sémantiques

Les deux approches sémantiques présentées ci-dessous prêtaient peu d'attention aux informations de types syntaxiques. Certains travaux de référence en sémantique ont cependant eu une approche plus souple: .... Le système de compréhension de textes KALYPSOS de Fargues [FARG 86], en cours de développement, utilise également un réseau sémantique inspiré des travaux de Sowa sur les graphes conceptuels. Son approche semble très complète et s'intéresse aussi bien aux aspects sémantiques des textes, que sémiotiques, morphologiques, syntaxiques.

1.2.4. Approches lexicales

Le rôle du lexique a longtemps été négligé par la communauté de TLN. Il suffit, pour s'en convaincre, de consulter le livre de référence de Winograd sur les théories syntaxiques représentatives en 1983 [WINO 83]. Pourtant, si l'on prend pour exemple le domaine de la génération, le problème du choix des mots, donc des relations entre les mots est primordial, comme nous l'avons signalé précédemment. Depuis quelques années, cette lacune tend à se combler [CL 87]. Plusieurs théories linguistiques, dont la dernière grammaire de Chomsky, la Grammaire Lexicale Fonctionnelle (LFG) ou la grammaire d'Unification Fonctionnelle (FUG) accordent une attention particulière à la structuration du lexique, aux relations ou fonctions entre les mots et à l'articulation entre la grammaire proprement dite et le lexique. Une partie des informations linguistiques, autrefois décrites dans la grammaire, le sont aujourd'hui au niveau du lexique. On a tendance à vouloir diminuer le nombre et le rôle des règles grammaticales (cf, entre autres, la grammaire catégorielle UCG[10]).

Nous évoquons ici deux théories linguistiques qui se sont intéressées très tôt au contenu et à l'organisation du lexique, au point d'en faire l'élément central de leurs approches: les Lexiques-Grammaires et la théorie Sens-Texte. La grammaire n'y apparaît plus comme une entité séparée du lexique mais disparaît complètement. Les relations grammaticales s'expriment au travers de la structuration du lexique et des fonctions lexicales (ces fonctions n'existent que dans la théorie Sens-Texte).

[10] LFG, FUG et UCG sont des grammaires d'unification. Des références précises sont données au chapitre 2.

1.2.4.1. Les Lexiques-Grammaires

Se démarquant nettement de l'approche chomskienne et syntagmatique, Gross ([GROS 75] [GROS 84]) refuse toute théorie formelle initiale du langage et tente d'inventorier systématiquement les différentes constructions syntaxiques du français. A la façon de l'approche distributionnelle d'Harris tout niveau d'abstraction n'est introduit que si les données l'y obligent. Mais il va même plus loin en rejetant l'idée de règles fondées sur les catégories grammaticales. D'après lui, le nombre de règles ne fait que croître en fonction de la taille du corpus et aucun phénomène de convergence n'apparaît. Il constate de plus qu'aucune théorie linguistique actuelle ne permet d'expliquer nombre d'expressions usuelles du langage. Sa démarche consiste alors à mener une étude approfondie sur les relations syntaxiques qui régissent chaque mot d'une phrase.

Gross et l'équipe du LADL ont ainsi recensé près de 12000 verbes du français et dégagé 500 propriétés syntaxiques: distributionnelles (structuration en sous-classe syntaxico/sémantiques) et transformationnelles (passivation, extraposition,...). Précisément, ce n'est pas le mot simple qui constitue l'unité du dictionnaire, mais la construction sujet-verbe-objet et ses propriétés syntaxiques. L'ensemble des items du dictionnaire se présente sous forme de table et constitue les lexiques-grammaires. Il faut noter également qu'un recensement des expressions figées et des mots composés est en cours.

Les lexiques-grammaires constituent donc une des sources de connaissances les plus importantes du français. Le problème se pose de savoir comment utiliser ces dictionnaires dans des applications portant sur des corpus importants. En génération [DANL 85] et en correction automatique, le travail a déjà commencé. En analyse peu de choses ont été faites. Lancel & al [LANC 88] ont couplé une approche lexique-grammaire couvrant les aspects syntaxiques avec une représentation sémantique sous forme de description fonctionnelles [KAY 82] dans un projet d'interface en anglais.

En fait le problème n'est pas vraiment abordé, car il n'existe pas de dictionnaire de l'anglais construit avec l'approche lexique-grammaire, et ce qui a surtout, nous semble-t-il, été utilisé dans ce projet sont les outils d'analyse distributionnelle classique. Le problème de la connexion, sur de larges corpus, des items lexicaux avec des concepts ou traits sémantiques restent entier. Sans doute parce qu'il n'existe pas aujourd'hui de grand projet de TLN comparable aux projets anglo-saxons. L'un des intérêts d'un tel projet serait de voir la part des connaissances sémantiques, connaissances du monde par rapport aux connaissances syntaxiques sur un domaine ouvert[11].

---

[11] Les noms/expressions composés représentent une part importante des corpus des domaines techniques. Sur un domaine technique particulier, des travaux ont montré la place importante de la sémantique dans l'explication de la formation de ces expressions. Dans le vocabulaire général de la langue, Gross [GROS 86] a montré la non compositionnalité d'une expression à partir du sens de ses mots simples et rejette l'importance de la sémantique : différence de point de vue ou différence d'échelle ?

1.2.4.2. La théorie Sens-Texte

La théorie Sens-Texte ([MELC 70], [MELC 87]), qui décrit un ensemble de modèles mettant en relation des textes[12] avec leurs sens linguistiques, est issue d'un travail descriptif sur les propriétés sémantiques, syntaxiques, morphologiques et lexicales du russe. Elle a été testée de façon détaillée sur le français et l'anglais.

Le lexique occupe une place centrale dans cette théorie qui s'attache à mettre en évidence le comportement propre de chaque lexème. A la différence du travail du LADL, la description lexicale met l'accent sur les relations entre lexèmes en termes sémantiques. Beringer présente ainsi la notion de représentation sémantique dans le Modèle Sens-Texte [BERI 88]:

> "Ici, la représentation sémantique est l'invariant canonique d'un ensemble d'énoncés synonymes... Si le linguiste cherche à valider l'intuition qu'il peut avoir sur la synonymie de deux énoncés, il vérifiera qu'ils sont substituables l'un par l'autre dans la majorité des discours sans en changer la signification."

Un système composé d'une soixantaine de fonctions lexicales permet de décrire la plus grande partie des phénomènes de co-occurrence entre mots que l'on retrouve dans les expressions idiomatiques d'une langue donnée.

Au niveau syntaxique, les relations grammaticales, comme *sujet*, *objet*,... sont directement représentées dans les arbres syntaxiques. Ces relations étiquettent les arcs reliant les noeuds lexicaux de ces arbres. La notion fondamentale d'unicité du sujet s'expriment ainsi plus simplement que dans les anciennes grammaires transformationnelles dans lesquelles on ne pouvait accéder qu'indirectement aux relations grammaticales.

Les règles lexicales traditionnelles se traduisent ici par la mise en relation d'items lexicaux au travers d'arbres de dépendances. Ces relations sont définies dans le lexique.

---

[12] Aujourd'hui la théorie se limite à l'étude des relations intra-phrases, même si, en principe, elle peut s'appliquer à des textes entiers.

La structure d'ensemble du Modèle Sens-Texte comprend sept niveaux de représentations et des règles de mise en correspondance bi-directionnelles entre chaque niveau. Toutes les propriétés linguistiques d'un texte comprises entre les niveaux phonologique et sémantique sont ainsi prises en compte (mais ne sont pas encore toutes détaillées avec la même précision). Des réseaux sémantiques sont utilisés pour représenter la décomposition du sens d'un énoncé sur des critères purement linguistiques[13]. Ces décompositions permettent de faire apparaître les "primitives sémantiques", propres à une langue et non pas universelles comme celles de Schank. Bien que les étiquettes relationnelles utilisées à chaque niveau soient très similaires d'une langue à l'autre, le vocabulaire des termes des arguments et le détail des mises en correspondance entre niveaux sont très dépendants de la langue étudiée.

Les travaux linguistiques dans le cadre la théorie Sens-Texte datent déjà d'une vingtaine d'année, mais les premières implantations informatiques sont, elles, très récentes: Beringer l'a appliquée, dans sa thèse, à des problèmes de désambiguïsation en français et l'équipe de Kittredge, à la génération de messages de systèmes d'exploitation informatiques [IORD 88].

1.2.5. Remarque

Cette présentation générale de différentes théories de la langue n'est nullement exhaustive. Outre les grammaires d'unification qui seront présentées dans le chapitre suivant, certaines grammaires importantes n'ont même pas été évoquées, bien qu'elles soient toujours utilisées et fassent l'objet de recherches : les grammaires systémiques [HALL 85], catégorielles, les grammaires d'arbres conjoints (*tree-adjoining grammar*) [JOSH 85]. Certains rapprochements ont été entrepris avec les grammaires d'unification.

Les grammaires de cas [FILL 68] ont profondément marqué la linguistique informatique, à l'époque de la toute puissante approche syntaxique chomskienne, en introduisant un certain nombre de cas sémantiques traduisant les relations entre les noms et le verbe dans une phrase simple. Aujourd'hui, elles ne font plus l'objet de développements particuliers (bien que les discussions sur le nombre et la nature des cas primitifs soient encore ouvertes [BOGU 87]), mais ces notions de cas sont généralement intégrés dans les autres grammaires.

---

[13] les critères liés au domaine d'un texte (au sens introduit dans la première partie de ce chapitre) ne sont pas traduits dans cette théorie.

Le formalisme des ATN (*Augmented Transition Network*) [WOOD 80] n'est pas cité ici. Ce n'est pas une théorie linguistique (il n'apporte pas d'éclairage spécifique sur le traitement du langage naturel), mais un outil linguistique qui a longtemps été considéré comme le formalisme informatique le plus efficace pour implanter des grammaires. Nous l'évoquerons dans le chapitre suivant.

Quant aux grammaires sémantiques [BURT 79], elles ont été utilisées pour des développements rapides d'interface dans des domaines très limités. Ce sont plutôt des procédés ad hoc de repérage de groupe de mots à fort contenu sémantique que de véritables grammaires prenant en compte des phénomènes linguistiques.

1.3. OÙ ALLONS-NOUS ?

En 1987, l'un des forums de la conférence *Theoretical Issues in Natural Language Processing*, TINLAP-3, qui rassemblait une large fraction des courants évoqués précédemment, s'intitulait: "Pourquoi la théorie du traitement du langage naturel a-t-elle fait si peu de progrès ?" (sous-entendu: en 9 ans, depuis la conférence TINLAP-2 de 1978).

La distance entre la recherche et les applications est toujours très importante. Très peu d'entreprises font des affaires à partir de systèmes de TLN. Les interfaces les mieux vendues (et elles sont peu nombreuses) sont celles conçues à partir des technologies de la fin des années 70. La tendance actuelle, dans ce domaine, est de restreindre l'utilisation du langage naturel en guidant par exemple l'utilisateur avec des menus "intelligents" au lieu d'autoriser une frappe libre. Quant aux systèmes de traitement automatique de textes, aucun ne fonctionne sur des sources de textes réels alimentées en continu (nous ne parlons pas ici des approches statistiques ou de recherches documentaires qui se révèlent efficaces pour des analyses de surface des textes). Ces systèmes se heurtent à l'insurmontable problème de la taille des connaissances à gérer sur de grands corpus (comme en représentation des connaissances).

Les difficultés en TLN ont sans doute été sous-estimées, comme elles l'avaient été une première fois pendant la période "euphorique" des années 60 qui ont marqué les débuts de l'IA et de la traduction automatique. Malgré la multiplicité des approches, aucune théorie grammaticale n'a vraiment convaincu. Les anciennes technologies étaient souvent mises de côté sans avoir été explorées véritablement en détail. On pourrait aussi citer (comme le font les chercheurs du domaine eux-mêmes) les méthodologies étroites de travail: focalisation sur des solutions particulières sans égard à leur compatibilité avec d'autres problèmes, ignorance des travaux antérieurs, discrétion sur les limites des résultats obtenus,...

Pourtant les recherches en TLN sont nombreuses et prometteuses. Voici les points qui nous paraissent devoir connaître les développements les plus importants:

- les recherches sur des problèmes spécifiques tels : le discours, le contexte, les intentions du locuteur, les ambiguïtés lexicales, les métonymies, la prise en compte des phrases mal formées.

- la construction de systèmes reprenant partiellement plusieurs théories, avec des contraintes d'implantation d'efficacité et l'utilisation de bons outils informatiques. Les systèmes de Carter ou le système PUNDIT sont à cet égard exemplaire. Avec de telles préoccupations le problème de l'architecture devient primordial. L'architecture général du système ACTES que nous présenterons plus loin est assez classique. Par contre, celle utilisée dans le module composé de résolution de référents est plus originale. Partant du fait qu'il n'existe aucune approche complète des anaphores, nous essayons de faire coopérer chaque module apportant des réponses partielles au moyen d'une architecture de tableau noir.

- le développement de formalismes grammaticaux ayant pour vocation d'être facilement informatisables et suffisamment généraux pour décrire des grammaires de nature différente, traduisant les données syntaxiques et sémantiques dans des structures uniformes manipulables à l'aide d'un nombre limité d'opérations. La description d'une telle approche est l'objet du chapitre suivant où nous introduisons les notions de base "unificatrices" des grammaires d'unification. C'est celle que nous avons adopté en développant le formalisme AVAG.

2. LES GRAMMAIRES D'UNIFICATION

"A useful way of thinking about unification, from the point of view of computational linguistics, at least, is to see it as merely a useful computational procedure with a well defined semantics and efficient implementations."

"A unification formalism provides a rational basis for the comparison of different grammatical theories."

Pulman S., forum : "Unification and the new grammatism" ; TINLAP 3, 1987.

2.1. INTRODUCTION

Une grande partie des formalismes syntaxiques actuels peuvent être caractérisés par les points suivants [GAZD 87]:

- (i) recherche d'une sémantique déclarative des notations utilisées.
- (ii) règle de réécriture hors-contexte.
- (iii) analyse guidée par la compatibilité entre structure de données plutôt que la réécriture de chaînes de caractères (seule opération sur les chaînes, la concaténation).
- (iv) ensemble de catégories définies à partir de structures de traits.
- (v) l'unification est la première opération permettant de combiner les informations syntaxiques.

Les points (iv) et (v) caractérisent les grammaires d'unification. Un rapide historique nous permettra de montrer le large éventail de formalismes couvert par ce terme, les motivations qui ont présidées à l'extension de ce courant. Nous présenterons ensuite les caractéristiques générales, structures de données et opérations, des grammaires d'unification en signalant quelques variantes de certains formalismes, les extensions sur lesquelles portent les recherches. Nous évoquerons enfin, l'approche de Aït-Kaci, différente de celle de la linguistique informatique, venant de l'IA, et qui utilise des structures de données et opérations voisines permettant de traduire dans un même formalisme des connaissances sémantiques.

L'intérêt de cette approche pour le traitement du langage naturel est la volonté de développer des formalismes grammaticaux à partir d'opérations et de structures déclaratives, relativement claires sémantiquement, se prêtant bien à des implantations efficaces, modulaires, d'autant que ces structures et opérations sont très voisines de celles couramment utilisées en programmation logique, et en particulier en PROLOG. La référence à des mécanismes bien connus en informatique théorique offre aussi une base pour comparer les différents formalismes grammaticaux.

2.2. BREF HISTORIQUE

La branche des grammaires d'unification [SHIE 86] est le résultat de la confluence de trois courants du traitement du langage naturel : un courant européen issu de la **programmation logique**, un courant nord-américain visant à développer des **outils d'informatique linguistique** et enfin un courant anglo-saxon développant des t**héories linguistiques formelles** (cf le schéma de la figure 2.1).

PROLOG ([GIAN 85], [SHAP 86]) se prête bien à l'écriture d'applications de traitement du langage naturel [PERE 87 b], pour au moins trois raisons: des grammaires partielles d'un langage naturel s'écrivent facilement et directement sous forme de programmes PROLOG. La proximité de ce langage informatique avec la logique des prédicats permet de donner aisément une représentation sémantique logique à des textes. Enfin le moteur PROLOG permet de gérer à bon compte des inférences sur ces structures sémantiques[1]. Alain Colmerauer et son équipe ont d'ailleurs développé PROLOG dans l'intention de l'appliquer en priorité au traitement du langage naturel. Dès 1973, il a développé un formalisme, les grammaires de métamorphoses [COLM 78], se traduisant facilement en code PROLOG. L'aspect sémantique de son approche du TLN est présenté dans [COLM 79].

Fernando Pereira a ensuite développé un aspect particulier des grammaires de métamorphoses sous le nom de grammaires à clauses définies (DCG). Sa simplicité et sa puissance d'expression comparable à celles des ATN [PERE 80] ont fait des DCG un formalisme grammatical couramment utilisé (nous le reprenons dans notre travail, cf la partie sur la compilation du formalisme AVAG). Bien d'autres **grammaires logiques** ont depuis lors été développées (cf [SABAT 87] pour une bonne introduction et [MCOR 87] pour un exemple détaillé d'utilisation en PROLOG). Ce qui nous intéresse particulièrement ici, c'est que toutes représentent les informations linguistiques dans des structures de traits qui sont les **termes PROLOG**, et traduisent les contraintes linguistiques par l'opération d'**unification de PROLOG**.

Un autre courant, nord-américain, a le souci de développer des formalismes grammaticaux se prêtant aisément à l'implantation et traduisant bien les façons de s'exprimer propres aux milieu linguistique informatique. Il est né des travaux de Martin Kay en 1979 ([KAY 79] et [KAY 82]). Sa grammaire, la **Functional Unification Grammar** (FUG), avait, à l'origine, pour but de clarifier et implanter la notion linguistique de trait, déjà utilisée dans les grammaires systémiques, certaines grammaires de Chomsky et certaines grammaires syntagmatiques. Alors que ces dernières grammaires n'utilisaient que des **structures de traits** simples (traits à valeurs uniquement atomiques, ou structures uniquement représentables par des arbres), Kay a construit des structures plus complexes, les descriptions fonctionnelles, représentables par des graphes et a défini une opération d'unification (sans référence à l'unification de la logique) opérant sur ces descriptions. Sa terminologie est à l'origine de l'expression 'grammaire d'unification'.

---

[1] Sous réserve que le nombre de règles ne soient pas trop important. Nous reviendrons, lors de la description de notre analyseur, sur la stratégie PROLOG dont l'efficacité est limitée.

Parallèlement Fernando Pereira et Stuart M. Shieber, entre autres, ont développé le formalisme PATR-II ([SHIE 83] et [SHIE 84]), dont AVAG s'inspire. Ils utilisaient également des structures de traits et une opération d'unification, couplés avec un système de règles de réécriture hors-contexte. Inspirés par les exemples de DCG, FUG et GPSG, ils ont repris une partie des mécanismes mis en jeu dans ces grammaires et montré que leur formalisme général permettait de compiler (partiellement) nombre des grammaires que nous allons évoquer maintenant.

Le troisième courant des grammaires d'unification est celui centré, non pas sur l'élaboration d'outils pour le TLN, mais de théories linguistiques générales. Impressionnés par la rigueur des travaux de Montague [MONT 74] sur la sémantique d'un sous-ensemble de l'anglais, ils ont réagi contre les travaux plutôt informels qui marquaient la syntaxe alors. En quête de simplicité, ils ont repris des formalismes grammaticaux pré-transformationnels tels les grammaires syntagmatiques et les grammaires catégorielles [BARH 64], en les enrichissant avec des méta-règles, des structures de traits (encore appelés structures **attribut-valeur** , et une opération d'unification, toutes définies formellement. Parmi les plus connus, citons: Generalized Phrase Structure Grammar (GPSG) [GAZD 85], Lexical Functional Grammar (LFG) [BRES 82], Head-driven Phrase Structure Grammar (HPSG) [POLL 87] et sa version japonaise JPSG [GUNJ 87], Categorial Unification Grammar (CUG) [USZK 86], Unification Categorial Grammar (UCG) [CALD 88a]. Signalons que ces différentes grammaires n'abordent pratiquement que les aspects syntaxiques du TLN.

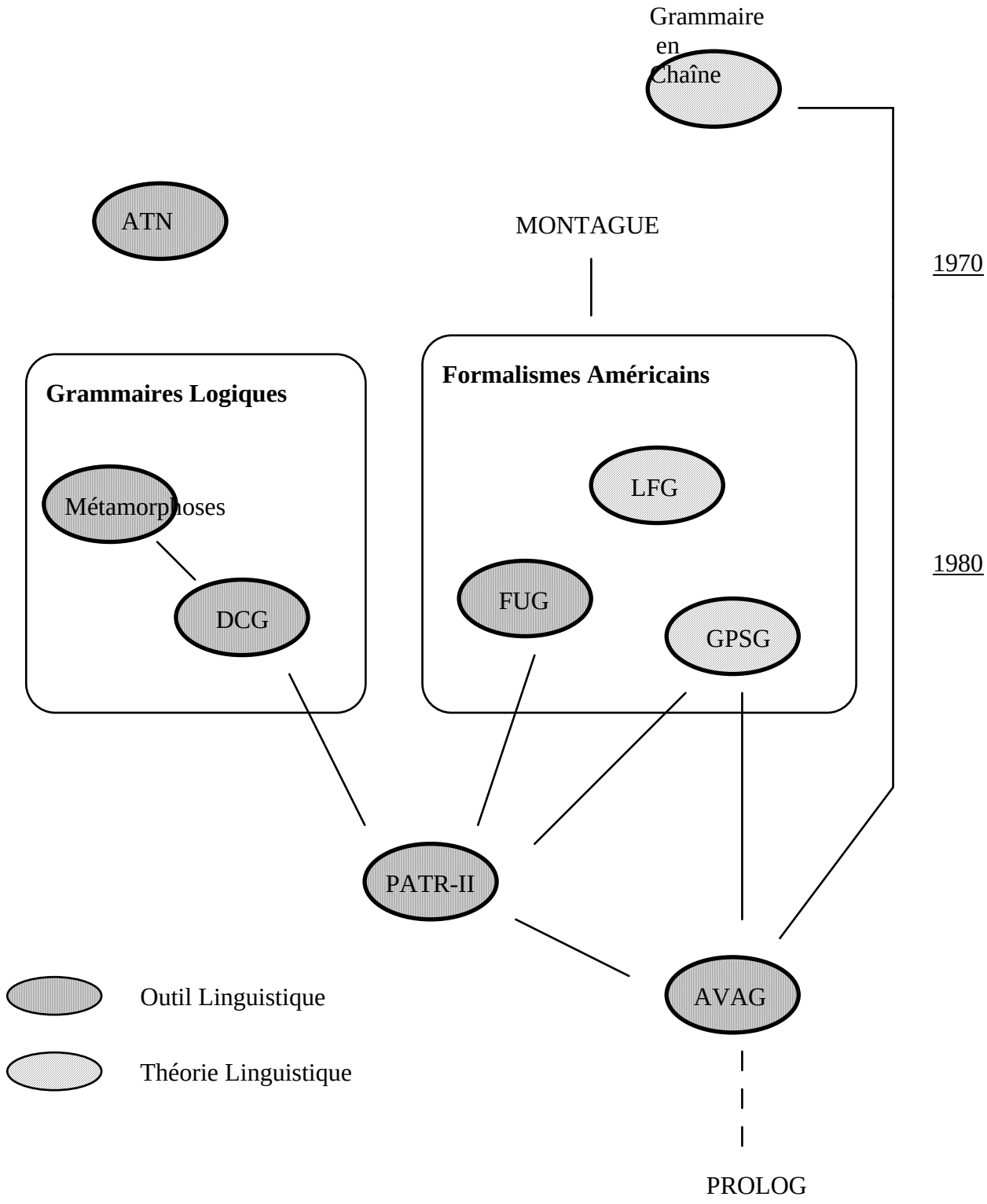


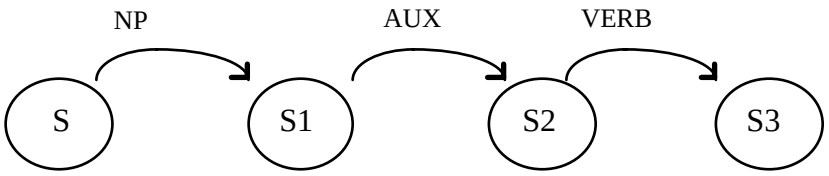
Fig 2.1: Les différents courants des grammaires d'unification ; influences sur AVAG

2.2.1. Motivations

Les préoccupations d'expressivité linguistique et déclarativité des formalismes ne sont pas les seules raisons qui expliquent l'extension des grammaires d'unification; Les chercheurs en linguistique informatique étaient également motivés par des problèmes procéduraux de TLN, en particulier par des **techniques d'analyse**.

Jusqu'à la fin des années 70, le modèle dominant utilisé pour écrire des grammaires sur ordinateur était celui des ATN (voir [WOOD 73] et [KAPL 73] pour les inventeurs et [BATE 78] pour une étude exhaustive), dans lequel informations morphologiques, syntaxiques, sémantiques, stratégies d'analyse étaient inextricablement liés. Il fallait pouvoir **séparer données et procédures** d'une part, mais aussi améliorer les procédures.

Un ATN peut se définir sommairement comme un réseau d'automates finis (cf figure 2.2): les noeuds représentent les états du processus d'analyse, les arcs sont étiquetés par les noms des catégories grammaticales. Un changement d'état implique l'analyse d'un groupe de mots correspondant à la catégorie grammaticale étiquetant l'arc correspondant. Des tests conditionnent ce changement d'état (accord en genre et nombre par exemple) et des actions suivent ce changement. Tests et actions sont décrits et implantés dans des registres. Or les informations exprimant les contraintes linguistiques ont une portée locale à ces registres, et sont réalisés par affectation de variables. La non-visibilité de ces informations et leurs assignations procédurales figent souvent les stratégies d'analyse et posent notamment problème en cas de réassignation des registres (cf [ALLE 87] pp 111-115, pour une discussion détaillée).



Arc	**Test**	**Actions**
S	--	SUBJ ← *
S1	--	AUX ← *
S2	if FORM* = en and $ROOT_{AUX}$ = BE	OBJ ← SUBJ SUBJ ← nil

Après l'analyse du groupe nominal(NP) son contenu est mis dans le registre SUBJ; même action après l'analyse de l'auxiliaire ; si l'auxiliaire est le verbe "be" et le suffixe du verbe est 'en' alors le contenu du registre sujet vient dans le registre objet , le registre sujet est réassigné à 'nil'.

Fig 2.2: Exemple de réassignation de registre dans un ATN pour une forme passive (d'après Allen)

Ces problèmes ont conduit des personnes comme Ronald Kaplan, qui avait contribué aux premiers développements des ATN, à utiliser l'unification en développant LFG. Dans les grammaires d'unification, la portée des informations est plus globale et l'assignations de traits à des valeurs est proche de l'instanciation de variables logiques et peut donc être facilement défaite en cas d'échec dans l'analyse. De plus les grammaires d'unification autorisent une description partielle des informations dans les structures de traits (d'où une économie dans la description et une garantie de cohérence au niveau de la grammaire) et le résultat de l'analyse est indépendant de l'ordre de description des règles. Ceci facilite l'implantation de stratégies variées.

2.3. STRUCTURES DE TRAITS ET UNIFICATION

Structures de traits et unification sont présentés à la façon du milieu linguistique informatique. Nous ferons à l'occasion quelques rapprochements avec l'approche Programmation Logique.

2.3.1. Categories grammaticales

Une catégorie grammaticale est (est associé à) un ensemble de spécifications de traits (ou structure de traits). Une spécification de traits est un couple ordonné attribut-valeur. L'attribut (ou trait) est atomique, la valeur est soit atomique soit complexe. Une valeur complexe correspond à une catégorie grammaticale. Ainsi dans la catégorie 'groupe nominal troisième personne du singulier' (notée D1), représentée sous forme de matrice dans la figure 2.3, les traits *cat* et *nombre* sont atomiques et le trait *accord* est complexe.

D1
$$\begin{bmatrix} \texttt{cat} = \texttt{GN} \\ \texttt{accord} = \begin{bmatrix} \texttt{genre} = \texttt{fem} \\ \texttt{nombre} = \texttt{sing} \end{bmatrix} \end{bmatrix}$$

fig 2.3: Structure de traits simple représentée sous forme matricielle

Cette même catégorie peut se représenter aussi sous forme d'arbre.

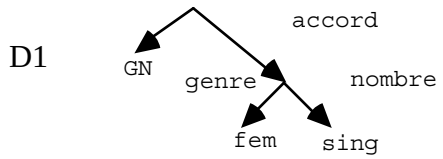


fig 2.4: Structure de traits sous forme d'arbre

L'ensemble de spécifications de traits définissant une catégorie satisfait certaines conditions. Tous les noms de traits ne figurent pas obligatoirement dans chaque catégorie, mais chaque occurrence d'un trait dans une catégorie n'est associé qu'à une seule valeur. Donc une catégorie peut être modélisée comme une fonction (et non une application) d'un ensemble de traits dans un ensemble de valeurs.

2.3.1.1. Subsomption

Si l'on note D(t), la valeur associée à t dans la structure de traits, alors D1(*nombre*) = *sing*, par exemple. Une structure de domaine vide est quelquefois appelée variable et notée : []. Une autre terminologie souvent utilisée est celle de chemin. Un chemin est une suite de traits. Voici la notation usuelle d'un chemin de D1 : <*accord nombre*>. Par extension du langage, pour désigner une sous-structure de D1, on pourra écrire D1(<*accord nombre*>) = *sing*.

Sur ces structures de traits, on peut définir une relation d'ordre partiel appelée **subsomption**[2]. Une structure D1 subsume D2 (ce que l'on note souvent D1 $\subseteq$ D2) si, informellement, D1 apporte moins d'informations que D2. Par exemple, dans la figure 2.5, la locution "une femme" est moins décrite par la structure D1 que par D2. D1 apporte moins d'informations.

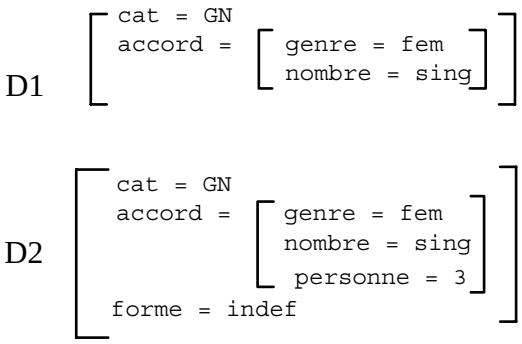


Fig 2.5: Subsomption et description d'informations

Plus précisément D1 $\subseteq$ D2, si D1 et D2 sont des structures atomiques identiques ou bien sont des fonctions dont le domaine de D1 est inclus dans celui de D2 et, pour tout trait t du premier domaine, alors D1(t) $\subseteq$ D2(t). Si D1 subsume D2, on dit aussi que D2 est une instance de D1[3].

---

[2] cf le terme "subsumer" utilisé en logique. La subsomption (*subsumption* en anglais) sera le substantif correspondant au verbe.

[3] A noter que l'on trouve aussi la terminologie suivante:au lieu de dire que D1 subsume D2, on dit que D2 est une extension de D1. On parle alors de la relation d'extension, au lieu de subsumption.

2.3.1.2. Unification

Deux catégories D1 et D2 sont dites consistantes s'il existe une catégorie D3 qui est subsumée par D1 et D2, donc D1 $\subseteq$ D3 et D2 $\subseteq$ D3.

Si deux catégories sont consistantes, il existe, dans l'ensemble des catégories subsumées par les deux premières, une catégorie qui subsume toutes les autres. C'est la borne supérieure de cet ensemble, notée D1 ≈ D2. Cette borne est l'**unification** de D1 et D2. Ainsi l'unification de D1 et D2, dans la figure 2.6 est D3, mais D1 et D4 ne s'unifient pas[4].

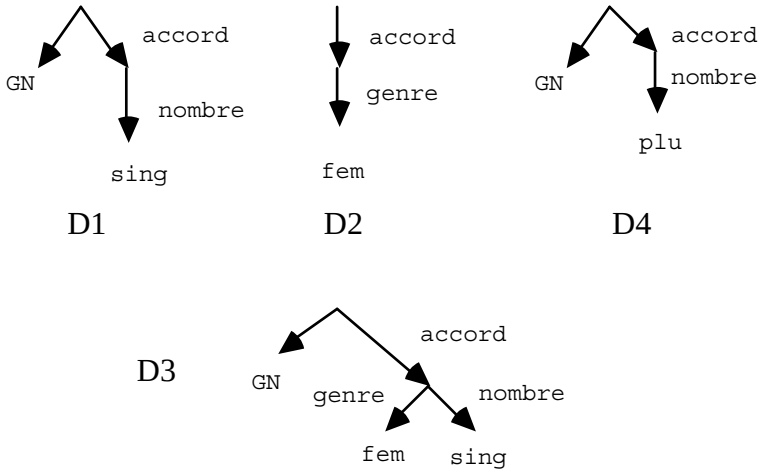


Fig 2.6: Unification sur des structures de traits

2.3.1.3. Graphes de traits

Les structures de traits représentables par des arbres (les arbres de traits) correspondent bien à la notion linguistique de catégorie. Mais les catégories construites dans les règles de grammaire sont souvent le résultat de l'application de contraintes sur des sous-arbres. Ces catégories se représentent alors sous forme de graphes de traits. La figure 2.7 représente un graphe D1, dans lequel sont exprimées les contraintes d'accord entre le sujet et le verbe. D3 est l'unification de D1 et D2.

---

[4] En programmation logique, D3 représente **le plus grand unificateur** de D1 et D2, et l'unification est l'opération qui calcule ce plus grand unificateur, puis remplace D1 et D2 par D3. Les grammaires d'unification opèrent souvent ce remplacement et l'appellent alors *unfication destructrice* ou *unification* tout court. Dans ce dernier cas, il y a alors confusion avec le terme d'unification définit précédemment.

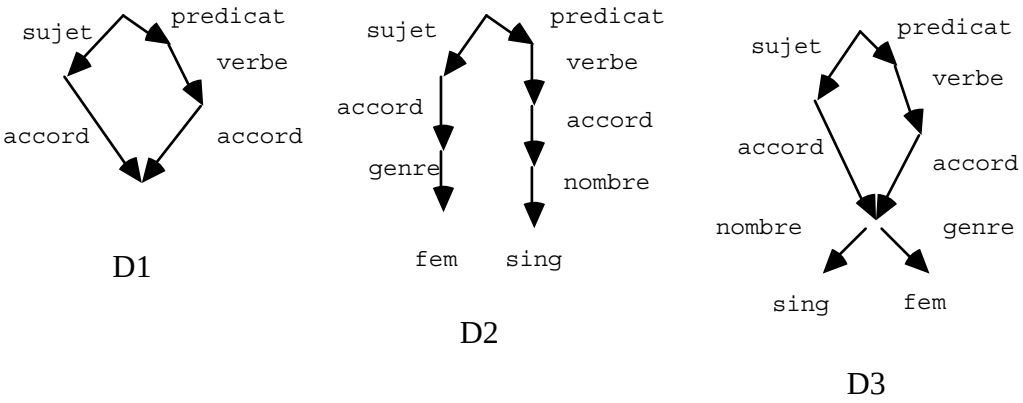


Fig 2.7 : Unification sur des graphes

Les graphes représentés sur cette figure sont des graphes orientés sans circuit, encore appelés DAG (pour *Directed Acyclic Graph*). Les graphes qui ne sont pas des arbres sont appelés graphes réentrants, ou à valeurs partagées (ici les valeur du trait *accord*).

Certaines grammaires décrivent certains phénomènes linguistiques qui s'expriment sous forme de graphe avec circuit. Le modèle de l'unification et des structures de traits présenté informellement précédemment doit alors être modifié. Des modèles d'automates d'états finis ou encore utilisant une sémantique dénotationnelle (pour PATR-II dans [PERE 84]) ont été proposés. Dans ce dernier, un graphe se décrit comme un ensemble d'équations. Ainsi D3 se représente :

```
{ < sujet accord > = < predicat verbe accord>, < sujet accord
nombre> = sing, < sujet accord genre> = fem }
```

2.3.2. Variantes

Les différentes grammaires d'unification n'utilisent pas toutes le même genre de structures de traits. Ainsi, FUG et PATR-II acceptent des DAGs, mais pas de graphes avec circuit. GPSG et LFG se contentent d'arbres de traits.

Des mêmes fonctionnalités se traduisent de façon fort différentes d'une grammaire à une autre. Ainsi, par exemple, la plupart des grammaires s'accordent généralement sur le fait que, dans une règle, les traits principaux (rassemblés sous le trait *head*) du constituant principal d'une queue de règle doivent être identiques à ceux de la tête de règle. Dans GPSG, cela est stipulé par la convention *head feature convention* qui s'applique à toute la grammaire. Dans PATR-II, cela s'exprime explicitement par une équation dans chaque règle de grammaire.: <*X0 head*> = < *X2 head*> dans l'exemple de la figure 2.8.

```
X0 -> X1 X2
          <X0 cat> = ph
          <X1 cat> = gn
          <X2 cat> = gv
          <X0 head> = <X2 head>
          <X0 head sujet > = < X1 head>
```

Fig 2.8 : Exemple de *head feature convention* traduit dans une règle PATR-II

La règle hors-contexte traduit l'opération de concaténation entre deux chaînes, l'une de catégorie groupe nominal GN, la seconde de catégorie groupe verbal GV. Le résultat est la chaîne de catégorie phrase PH, sous réserve que les contraintes définies dans les équations soient satisfaites. Ces équations stipulent les unifications entre les structures de traits associées à chacun des constituants de la règle. En particulier les traits principaux du groupe verbal, rassemblés sous le trait *head*, doivent être identiques à ceux de la phrase.

De nombreuses grammaires ont des mécanismes de valeur par défaut pour les traits. Dans GPSG, par exemple, les *Feature cooccurrence restrictions* expriment des conditions impératives sur les valeurs de certains traits. Les *Feature specification defaults* stipulent des valeurs par défaut non obligatoires. Dans FUG, la valeur *ANY* donnée à un trait spécifie que ce trait devra être instancié au cours de l'analyse à une valeur quelconque. Certains de ces mécanismes sont tout à fait procéduraux et ne sont décrits par aucun modèle.

2.3.3. Comparaisons avec PROLOG

La similitude entre les structures de données et opérations des grammaires d'unification et celles de PROLOG est très grande. Structures de DAGs et termes PROLOG se correspondent assez bien, unification présentée précédemment et unification PROLOG aussi. Les valeurs partagées dans les structures de traits, par exemple, s'expriment aisément par l'unification d'arguments dans des termes PROLOG. Quant aux graphes avec circuit, c'est un problème que PROLOG-II sait traiter depuis longtemps avec l'unification sur les structures d'arbres infinis. Colmerauer a proposé [COLM 83] un modèle correspondant en utilisant sa propre terminologie: ses structures sont des arbres relationnels ; il ne parle pas d'instanciation, ni d'unification, mais d'affectation sylvestre et d'ensemble de contraintes s'exprimant par des équations ou inéquations sur les arbres relationnels. Une présentation, plus proche de la terminologie Programmation Logique usuelle peut être trouvée dans [JAFF 84].

Ceci dit, des différences substantielles existent entre grammaires d'unification et formalismes grammaticaux très proches de PROLOG, tels les DCG. La notion de **description partielle** est fondamentale en linguistique informatique. On peut classer une locution comme appartenant à une catégorie donnée en utilisant seulement une description partielle de la structure de traits de cette catégorie.

Prenons l'exemple des DCG. Dans ce formalisme, les catégories sont représentées par des termes composés dans lesquels les arguments jouent le rôle de traits. La règle PATR-II de la figure 2.8 peut se représenter ainsi en DCG:

```
ph(head(SujetHead,Forme)) -->
          gn(SujetHead),
          gv(head(SujetHead,Forme)).

Les contraintes d'unification définies dans les équations de PATR-II
sont représentées ici par l'identité (donc l'unification) des noms
des arguments des termes.
```

Fig 2.9 : Une règle DCG correspondante à la règle PATR-II de la figure 2.8.

Cette façon de représenter les traits n'est pas très modulaire. Si l'on désire ajouter un nouveau trait à une catégorie cela oblige à changer toutes les règles de grammaire correspondantes. Il faut de plus faire attention à l'ordre de description des traits dans une DCG. Dans la plupart des grammaires d'unification un trait d'une catégorie n'est présent dans une règle que si l'on y fait explicitement allusion. L'ordre des traits dans une structure est de plus indifférent[5].

A cette façon différente d'appréhender la notion de trait, s'ajoutent aussi tous les facilités d'expression de conditions sur les valeurs des traits évoquées dans le paragraphe précédent sur 'les variantes'.

2.4. EXTENSIONS

Nous présentons quelques extensions aux formes des structures de traits utilisées dans certaines grammaires et quelques opérations supplémentaires sur les structures de traits existantes dans quelques versions de grammaires. Toutes accroissent sensiblement le pouvoir d'expression des contraintes linguistiques. Les contraintes hiérarchiques et problèmes d'héritages liés seront évoqués plus loin lors de la présentation des travaux d'Aït-Kaci.

2.4.1. Descriptions disjonctives de traits

Il est intéressant de pouvoir regrouper les multiples descriptions associées à une entrée lexicale dans une même structure de traits. La figure 2.10 donne un exemple de description du mot "porte" en tant que verbe ou nom. Les parenthèses droites indiquent, comme précédemment, la conjonction et les accolades, la disjonction.

[5] Nous revenons sur ces problèmes dans notre présentation du formalisme AVAG qui est compilé en DCG.

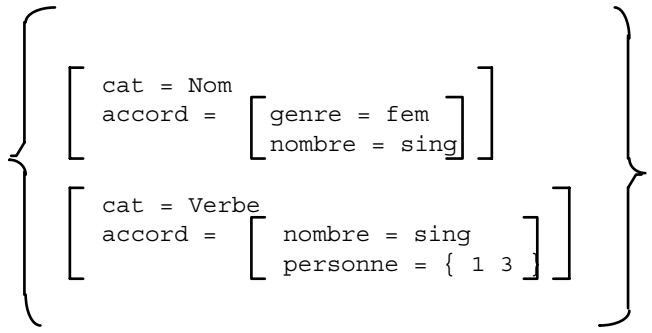


Fig 2.10 : Structure de traits avec valeurs disjonctives associée au mot "porte".

La disjonction est également intéressante dans la description des connaissances morphologiques [KART 84]. FUG l'utilise abondamment notamment pour factoriser l'écriture des règles grammaticales qui, dans ce formalisme ne se distingue pas des structures de traits (appelées *descriptions fonctionnelles* dans sa terminologie).

Les descriptions disjonctives posent plusieurs problèmes. Il faut trouver un nouveau modèle pour décrire ces structures et l'unification associée. Il faut également disposer d'algorithmes efficaces pour calculer l'unification de deux structures.

Une méthode consisterait à traduire une description utilisant des disjonctions sous forme normale disjonctive. Ainsi dans une DCG, lorsque l'on veut éliminer des disjonctions dans les termes, il suffit de réécrire la règle contenant les disjonctions en une suite de règles alternatives. FUG procède également par décomposition. Mais cette méthode conduit rapidement à une explosion combinatoire en temps de calcul et en place mémoire dès que la grammaire atteint une taille significative.

Kasper propose une autre approche. Il présente un nouveau modèle pour décrire les structures de traits ([KASP 88] et [KASP 86] pour une présentation linguistique, [ROUN 86] et [MOSH 87] pour une description formelle détaillée). Une structure est représentée par une formule logique. Ainsi l'exemple de la figure 2.10 peut se traduire par:

```
porte = (cat: Nom ∧ accord: (nombre: sing ∧ genre: fem))
 ∨ (cat: Verbe ∧ accord: (nombre: sing ∧ personne: ( 1 ∨ 3)))
```

Bien que le problème de l'unification de descriptions disjonctives soit NP-complet et qu'il n'existe donc pas d'algorithme général de complexité polynomiale, Kasper propose un algorithme évitant d'avoir recours à la décomposition systématique [KASP 87] (voir aussi [EISE 88] pour une version améliorée). Sa stratégie consiste à unifier d'abord les parties complètement définies des structures et différer l'analyse des parties non complètement instanciées. Les disjonctions sont éliminées d'une description dès qu'elles sont inconsistantes avec des informations définies.

2.4.2. Généralisation et contraintes négatives

Karttunen a montré [KART 84] l'intérêt de l'utilisation de la généralisation et des contraintes négatives pour exprimer des connaissances morphologiques ou les contraintes d'accord entre deux locutions conjointes et la coordination de structures de traits[6].

Informellement, alors que l'unification évoque l'union de deux ensembles de traits, la **généralisation** correspond à l'intersection de ces ensembles. Ainsi, dans la figure 2.11, D3 est la généralisation de D1 et D2.

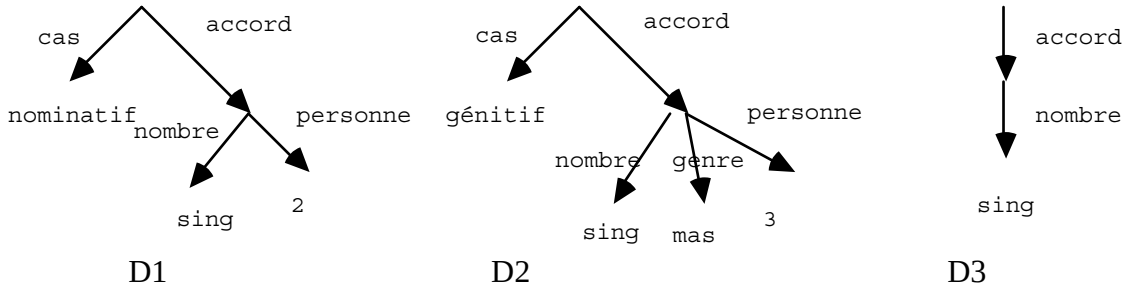


Fig. 2.11 : Généralisation de deux structures

2.4.2.1. La négation

La négation permet d'exprimer des contraintes linguistiques intéressantes. Ainsi la contrainte indiquant que, dans une locution, sujet et objet ne peuvent référer à la même entité, à moins que l'objet ne soit un pronom réfléchi, peut se représenter par cette formule logique à la Kasper:

```
¬ (obj: refl:-
          ∧ { <subj ref> , <obj ref> } )
```

La notation entre accolades correspond à l'identité des valeurs des deux chemins <*subj ref*> et <*obj ref*>.

La description partielle des structures de traits posent quelques problèmes pour l'interprétation de la négation. Si l'on retient la façon classique d'interpréter la négation, on dira qu'une description satisfait la négation d'une formule ssi elle ne satisfait pas la formule positive:

$(D \models \neg f) \quad <-> \quad (D \not\models f)$

Mais alors nous avons le comportement bizarre illustré par l'exemple suivant: dans la figure 2.12, D1 satisfait la formule suivante, mais pas son instance D2:

---

[6] Nous présenterons notre propre version de la généralisation appliquée aux problèmes de coordination, ainsi que quelques contraintes négatives simples avec la description du formalisme AVAG.

accord: ¬ (genre:fem ∧ nombre: sing)

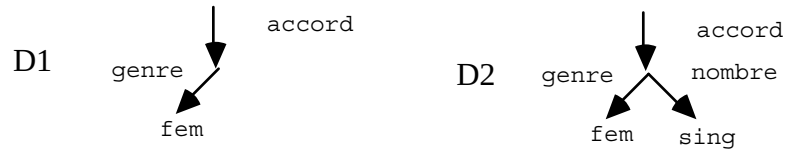


Fig 2.12: Une structure et son instance

Nous pouvons convenir alors qu'une structure satisfait ¬ *f* ssi aucune de ses instances ne satisfont *f* .Cependant une telle proposition a pour conséquence que des structures un peu complexes satisfont difficilement des formules négatives, comme le montre l'exemple suivant de Pereira. La structure D1 de la figure 2.13 ne satisfait pas la formule

¬ { < sujet accord>, < predicat verbe accord> }

comme le montre son instance D2, mais elle ne satisfait pas non plus

¬ ¬ { < sujet accord>, < predicat verbe accord> }

comme le montre son instance D3.

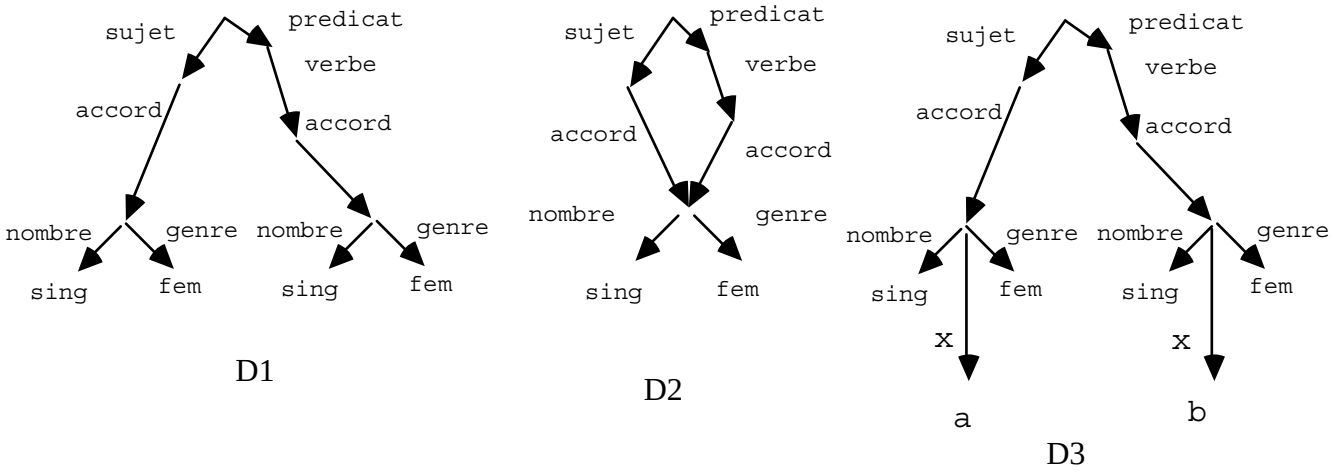


Fig 2.13: Une structure et des instances incompatibles

Pour éviter ce genre de problème, il faudrait pouvoir restreindre l'ensemble des instances possibles d'une structure, en typant les traits par exemple.

Cette façon d'aborder le traitement de la négation est très proche du traitement des inégalités sur les arbres dans Prolog-II. Le *dif* Prolog retarde l'exécution d'une contrainte d'inégalité entre deux termes t1 et t2 jusqu'à ce qu'ils soient suffisamment instanciés pour pouvoir vérifier l'inégalité ou pour la remplacer par une disjonction d'inégalités sur les sous-arbres. Dans le modèle de Prolog-II, deux termes sont différents s'ils n'ont pas d'instance commune. On ne peut toutefois traduire la négation des grammaires d'unification en contraintes d'inégalités en Prolog, puisque dans les grammaires d'unification les arbres n'ont pas une arité fixe. Nous avions déjà fait la même remarque à propos de l'unification (cf une réponse à ces problèmes est proposée au chapitre 6 avec la compilation du formalisme).

2.5. TYPES ET REPRESENTATION DES CONNAISSANCES

La structure des couples attribut-valeur, constituant les structures de traits, présentées jusqu'à présent était relativement libre. Un nombre grandissant de grammaires d'unification tend à imposer des contraintes sur les traits ou les valeurs des traits de façon à mieux organiser les connaissances et disposer de procédures de calcul efficaces et de vérification de cohérence sur ces connaissances. Nous présentons deux tendances différentes l'une ayant pour finalité l'organisation de connaissances purement linguistiques, l'autre se prêtant bien à la représentation des connaissances du domaine d'un texte. Toutes deux s'appuient sur la notion de type (comprise dans des sens différents) et permettent d'exprimer des contraintes hiérarchiques.

2.5.1. Types et sorts dans UCG

Une structure de traits dans la grammaire UCG [CALD 88a] est constituée de huit traits typés. En voici sa définition générale [CALD 88 b]:

```
declare(structure_de_traits,
          phonologie ** {listephon, atome}:
          catégorie ** {complexe, cat_base, atome}:
          sémantique ** {var_inl, sem, liste_occ}:
          ordre:
          in ** traits_environnement:
          out ** traits_environnement:
          liste_occur ** {atome, liste_occ, impropre}:
          nombre_occur).
```

Chaque trait est défini par son étiquette et son type de valeur: *étiquette ** type*. Quand le type est absent c'est que par défaut le trait correspondant ne peut être instancié qu'à une valeur atomique (*atome*) au sens de Prolog. Les accolades expriment une disjonction sur les types de valeurs possibles pour un trait. Ainsi, le trait *sémantique* a des valeurs de type *var_inl* ou *sem* ou *liste_occ*, c'est-à-dire qu'il peut représenter une variable INL du langage de représentation sémantique (cf la notion de référent du discours dans la présentation de la DRT) ou une structure sémantique ou une liste d'occurrences dans laquelle sont stockées les informations syntaxico-sémantiques sur le référent du discours dénoter par une variable INL (informations utilisées lors de la recherche de référents). Chaque type doit être défini à son tour. Voici la définition du type *sem*:

```
declare(sem, index ** var_inl:
          prédicat ** { atome, liste, traits_environnement}:
          liste_arg ** {atome, argsem }).
```

Une structure sémantique est donc composée d'un index, d'un prédicat et d'une liste d'arguments. L'index joue un rôle particulier: son type var_inl identifie la variable introduite et son **sort**. Ce sort sont nous allons reparlé classe toute la structure sémantique. Si un référent du discours (variable INL) a été introduit par le mot "homme", l'index sera de sort *homme* et cette information conditionnera toute la structure sémantique.

Cette façon de déclarer explicitement tous les traits utilisés, leurs structures, permet au linguiste de bien classer les connaissances utilisées, d'attacher des structures de données particulières à chaque type de connaissances. Lors de la compilation de la grammaire (le formalisme UCG est compilé en code Prolog) des vérifications peuvent être faites et des incohérences dans l'utilisation des traits signalées à l'utilisateur.

Voyons maintenant, ce qu'il en est des connaissances représentées par ces sorts. Les auteurs ne sont concernés que par les connaissances linguistiques donc syntaxiques et se défendent de vouloir représenter les connaissances du domaine d'un texte. Ces connaissances sont définies ailleurs dans leur système, comme nous le verrons dans un autre chapitre. Un nombre limité de propriétés leur permet donc d'encoder ces connaissances. Un sort est construit à partir de propriétés définies exhaustivement:

```
propriétés([abstrait, évènement, féminin, générique, humain, mass,
mesure, neutre, singulier, temporel, fixe,...]).
```

Ainsi le mot "tomate" du lexique se verra attribué le sort de nom 'Neutre', ainsi défini:

```
def_sort('Neutre',[-temporel, -humain, +neutre, +singulier,-masse]).
```

Ce qui peut se comprendre par le fait qu'une tomate n'a pas d'aspect temporel, ni humain, qu'elle est neutre et singulier, isolable donc que l'ensemble des tomates est dénombrable (*-mass*). Le mot "vin", lui, sera de sort 'Masse' ([-temporel, +singulier,+mass]). Un tel classement permettra d'accepter ou refuser les expressions suivantes:

```
un litre de vin
* un litre de tomate
```

Des axiomes permettent d'établir des relations entre ces sorts dans le cadre d'une logique propositionnelle, comme, par exemple:

```
féminin → humain
humain → -neutre
singulier ∨ masse → -temporel
```

Un objet de sort *feminin* est donc *humain* et n'est pas *neutre*.

Sur l'ensemble des sorts on peut définir une relation d'ordre partielle, la subsomption, ainsi que l'unification et la généralisation de façon semblable à celle vue à propos de la définition des structures de traits au début du chapitre. Ainsi le sort (1) subsume (3) et (3) est l'unification de (1) et (2).

```
(1)       [-temporel, +humain, +singulier]
(2)       [-féminin]
(3)       [-temporel, +humain, +singulier, -féminin]
```

Pour calculer unification et généralisation entre sorts, les auteurs se ramènent au calcul d'intersection et d'union sur des ensembles de modèles. On définit un modèle en assignant des valeurs de vérité aux propriétés. Ainsi *+humain* signifie qu'à la propriété *humain* on assigne la valeur *vraie*. Pour n propriétés, il existe 2^n modèles, mais compte tenu des axiomes le nombre de modèles décroît fortement. Tous les modèles sont donc calculés explicitement et implantés comme des termes Prolog, en suivant l'idée proposée par Mellish pour encoder la classification des grammaires systémiques par unification dans [MELL 88].

2.5.2. Types et héritage dans LIFE

Les travaux de Aït-Kaci sont souvent cités (mais peu utilisés) dans le milieu des grammaires d'unification car il a construit une extension de Prolog [AITK 86] opérant sur des termes, les ψ-termes, très proches des structures de traits utilisées dans ces grammaires et disposant, en plus, d'un mécanisme de typage et de gestion des contraintes hiérarchiques. Dans un de ces derniers articles [AITK 89], il présente une nouvelle extension de son langage, LIFE, intégrant une partie des notions de la programmation fonctionnelle et surtout, pour ce qui nous concerne directement, une première application de LIFE à la manipulation de connaissances nécessaires dans le traitement du langage naturel.

Voici, par exemple, la description, à l'aide d'un ψ-terme, d'un adulte dans un conte de fée :

```
adulte(connait => adulte(connait => sorcière);
         hait => personne(connait => X: monarque;
                          aime => X))
```

On distingue trois types de symboles dans un ψ-terme: les variables (*X*), les traits (*connait*, *aime* et *hait*.), les types (*adulte, sorcière, monarque, personne*). Les variables sont toujours typées (par facilité syntaxique le type de *X* n'est pas répété ici). Une signature reflète les contraintes hiérarchiques entre les types, comme dans la figure 2.14

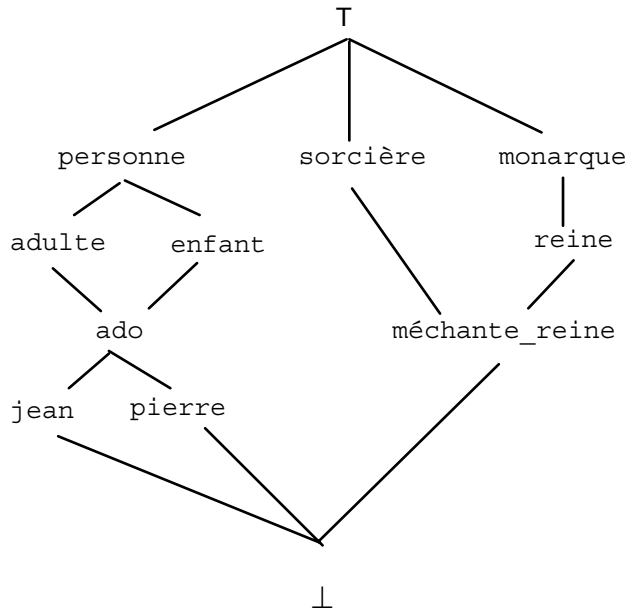


Fig 2.14: Une signature reflètant les contraintes hiérarchiques entre types

Ces relations hiérarchiques se déclarent à l'aide de l'opérateur '<' symbolisant la relation *est_un.*

```
ado < adulte.
ado < enfant.
enfant < personne.
{jean, pierre} < ado.
...
```

Comme sur les structures de traits (ou les termes de la logique du premier ordre), on définit une relation d'ordre partiel, la subsomption, puis l'unification et la généralisation. Ainsi l'unification du terme de type principal *adulte*, défini ci-dessus, s'unifie avec le terme suivant:

```
enfant(connait => X: personne(connait => reine;
                                 hait => Y: monarque);
          hait => enfant (connait => Y;
                          aime => méchante_reine);
          aime => X)
```

et donne:

```
ado(connait => X: adulte(connait => méchante_reine;
                         hait => Y:méchante_reine);
    hait => enfant(connait => Y;
                   aime => Y);
    aime => X).
```

La généralisation des deux donne:

```
personne(connait => personne;
          hait => personne (connait => monarque);
                            aime => monarque))
```

L'algorithme de calcul de l'unification de deux ψ-termes est détaillé dans [AITK 86] ([SMOL 89] pour une description formelle). Il opère sur les termes représentés sous forme de graphes orientés.

L'intérêt de l'application de LIFE au traitement du langage naturel est évident. On peut aisément exprimer des contraintes syntaxico-sémantiques, représenter à l'aide des types certaines connaissances sur le domaine d'un texte. Les notions de description partielle et non ordonnancement des traits sont toujours applicables.

Voici un extrait de grammaire simple dans laquelle on exprime des contraintes d'accord et des cas simples permettant de rejeter l'analyse de "une balle jette jean".

```
ph :- gn(nombre => N, classe => X),
          gv(nombre =>N, sujet => X).
...
dico(jette, verbe(nombre => singulier,
                  objet => projectile,
                  sujet => animé) ).
dico(jean, nomp(nombre => singulier, classe => humain).
dico(balle, nom(nombre => singulier, classe => projectile).
...
humain < animé.
```

Ceci dit, si l'expressivité d'un tel langage est attrayante, son efficacité est loin d'être évidente, comme le reconnaissent ses auteurs. LIFE est en effet écrit en Prolog, pour l'instant. Ecrire l'interpréteur d'un langage au dessus de l'interpréteur Prolog en utilisant un autre algorithme d'unification que l'algorithme de base de Prolog est une solution pratique mais très coûteuse au niveau des temps d'exécution.

2.6. DISCUSSION

Les grammaires d'unification nous semblent intéressantes à plusieurs points de vue. Les auteurs de ces grammaires cherchent des structures de données pour encoder les connaissances nécessaires au traitement du langage naturel et des opérations pour les manipuler, d'un haut niveau de déclarativité. Ils veulent élaborer des grammaires à partir de modèles ayant une sémantique claire et ce avec un double objectif: trouver une sémantique aux entités de base qui sous-tendent les structures du langage, comme les catégories ; construire des modèles qui se prêtent bien à une implantation efficace. Pour la première fois, une partie des linguistes, des linguistes-informaticiens qui ont participé aux premiers développements en TLN et ont développé, implanté des grammaires aussi différentes que les grammaires catégorielles, systémiques, ATN, syntagmatiques,.... se retrouvent dans un même courant unificateur ! Cette affirmation nécessite toutefois d'être nuancée.

Présentés avec des terminologies voisines, on assiste à un foisonnement de modèles Malgré une volonté de clarification affichée par chacun, il est souvent bien difficile de déceler les motivations des différents auteurs, les fonctionnalités réelles de leur système.

Il est certain que dans leur façon de présenter structures de traits et unification, les linguistes n'ont fait que redécouvrir, sans vouloir le reconnaître, des notions familières en programmation logique et ignorer les modèles développés en informatique théorique, comme par exemple celui de Prolog-II. Cela est sans doute dû à la coupure, effective il y a cinq ans encore, entre le milieu programmation logique (pour ne pas dire informatique) et le milieu des chercheurs en théories des grammaires, et la coupure entre le milieu "LISPien" d'Amérique du Nord, comprenant les chercheurs qui ont développé FUG et LFG, et le milieu "Prologien" européen.

Il ne serait pas juste pour autant d'en conclure que les grammaires d'unification ne sont qu'un "sucre syntaxique" au dessus de Prolog. Nous avons évoqué les nombreux phénomènes que les linguistes cherchent à exprimer directement dans leur formalisme. Ils ne peuvent être pris en compte dans le modèle Prolog standard. Il aurait été par contre intéressant de partir de modèles semblables à ceux de Prolog et de les étendre pour les besoins du traitement du langage naturel. Cette idée est défendue (plus ou moins explicitement) dans l'article de Pereira [PERE 87a] présenté à une conférence en programmation logique et dont nous nous sommes inspiré dans cette présentation. En retour, un tel rapprochement nous ferait progresser dans la caractérisation des différences entre langage naturel et langage artificiel.

Un autre problème, que nous avons cherché à illustrer dans ce chapitre, est celui de la **caractérisation des connaissances nécessaires au traitement du langage naturel**. Les auteurs de la grammaire UCG défendent ouvertement l'idée qu'il existe des connaissances "purement" linguistiques, isolables et qu'il est possible de ne s'intéresser qu'à ce type de connaissance dans une grammaire. L'idée n'est pas nouvelle (et n'est en rien singulière dans le milieu des grammaires d'unification). Nous l'avions évoquée dans le premier chapitre. C'est celle issue du courant des grammaires génératives. Elle est en relation avec l'idée de vouloir construire des grammaires générales de la langue, déconnectées des connaissances d'un domaine particulier, cherchant à caractériser les phrases "correctes" du langage.

Le phénomène remarquable ici c'est que cet a priori est en rapport direct avec la formalisation des structures manipulées (structures de traits, types sorts) dans la grammaire et les choix d'implantation, alors que la discussion avec les approches plus "informatiques" des grammaires d'unification se situe à un niveau très général[7]. Que deviendrait cette méthode de calcul de tous les modèles possibles, si l'on désirait décrire les connaissances d'un domaine sous forme de types ou sorts ?

A notre avis la distinction entre connaissances linguistiques-syntaxiques, d'une part, et connaissances sémantiques ou liées à un domaine particulier, d'autre part, n'est pas claire. Elle l'est d'autant moins lorsque le langage à traiter est celui d'un domaine restreint (cf l'approche des sous-langages dans le chapitre 1). Des régularités du langage sont facilement décelables quand il s'agit d'un "jargon" technique, ce qui est le cas dans notre projet. Les connaissances permettant de les analyser doivent pouvoir être mises en jeu le plus tôt possible lors de l'analyse. Peu importe de savoir si l'on encode des connaissances syntaxiques ou sémantiques . L'essentiel est de disposer de structures et mécanismes homogènes et suffisamment riches pour pouvoir les exprimer.

Le formalisme présenté par Aït-Kaci est intéressant pour modéliser des connaissances diverses. Mais il est encore limité pour permettre de décrire correctement les connaissances d'un domaine. Les types et termes peuvent servir à donner des descriptions des concepts d'un domaine. Mais les relations entre concepts concourent à affiner leur description. Les simples relations hiérarchiques décrites par l'opérateur '<' n'ont pas la richesse des rôles des langages "à la KL-ONE" ou des descriptions logiques comme celles du langage OMEGA [ATTA 81].

---

[7] on peut faire un rapprochement avec la démarche adoptée par Gazdar & al. dans [GAZD 88]. Dans cet article, ils présentent une formalisation des catégories grammaticales construit à partir de structures de traits et des opérations d'unification et subsomption. Avec ce formalisme, ils peuvent décrire les notions de catégories utilisées dans nombre de théories grammaticales (Chomsky, systémiques, catégorielles, ...) ... sauf celles des deux grammaires d'unification que l'on a présentées comme des outils informatiques, à savoir FUG et PATR-II !

Le problème est difficile. Vouloir tout intégrer dans un même formalisme est pour l'instant hors de portée. On connait l'efficacité informatique limitée de ces langages de représentation des connaissances.

A défaut d'une réponse à ce problème, nous avons essayé de connecter notre formalisme grammatical avec un langage 'à la KL-ONE', BACK, en introduisant une notion de typage correspondant aux concepts du réseau. L'intérêt est de pouvoir ordonner les descriptions des concepts de notre domaine en utilisant le classificateur du réseau et non 'à la main' comme dans LIFE. Par contre nous ne sommes pas en mesure, en l'état de nos développements, de récupérer et gérer la totalité des connaissances décrites dans le réseau.

3. SEMANTIQUE, TEXTE ET ANAPHORES

"There is in my opinion no important theoritical difference between natural language and the artificial languages of logicians. "

"The basic aim of semantics is to characterize the notions of a true sentence and of entailment"

Montague R.: "Universal Grammar", *Formal Philosophy*, 1973.

"En Informatique, on utilise couramment la notion de "contraintes d'intégrité"... Si l'on disposait de semblables contraintes dans le domaine du langage, on pourrait répondre affirmativement à la question [de savoir si l'on dispose de moyens rigoureux pour déterminer si une suite de mots a un sens] ; or il n'est pas nécessaire de chercher très loin pour voir que le langage ne respecte absolument pas les contraintes de ce type."

Kayser D.: "Une sémantique qui n'a pas de sens", *Langages* ; numéro spécial sur sémantique et Intelligence Artificielle, 1987.

"The strongest argument for DRSs as knowledge representations relate to their systematic connection with natural language...

Kamp H.: "Discourse Representation Theory: What it is and where it ought to go", *Natural Language at the Computer* , 1988.

"Since hearers do not have privileged access to a speaker's mind, other than through what a speaker says, imposing structure on the speaker's discourse will provide a framework for establishing the interpretation of anaphors."

Sidner C.L.: "Focusing in the Comprehension of Definite Anaphora", *Computational Models of Discourse*, 1983.

3.1. INTRODUCTION

Dans la partie précédente, nous nous sommes préoccupé de la description des aspects syntaxiques du langage dans les grammaires d'unification. Nous abordons maintenant le problème de la représentation intermédiaire du sens d'un énoncé avec deux séries de contraintes.

Le formalisme choisi pour la représentation sémantique doit s'intégrer facilement dans l'approche grammaire d'unification. Nous développerons ce point lors de la présentation des aspects syntaxiques et sémantiques du formalisme AVAG. D'autre part, il doit nous permettre de donner une représentation intermédiaire du sens d'un texte et pas seulement de phrases isolées. Etant donné la nature des textes à traiter dans le projet ACTES, cette représentation doit être assez complète (cf chapitre 1), c'est à dire prendre en compte le domaine et une partie du contexte afin de permettre de résoudre les nombreuses anaphores.

Dans un premier temps, nous allons préciser la place que nous attribuons à la sémantique dans notre approche. Nous espérons ainsi expliciter quelques sous-entendus et éviter les confusions entre logique et langage naturel trop souvent rencontrées dans les théories sémantiques formelles,comme la DRT (le chapitre 4 illustrera ce point). Puis nous présenterons la DRT telle que définie par Kamp en 1981. Dans la dernière partie, nous nous intéresserons au problème de la résolution des anaphores et discuterons des apports et limites de la DRT sur ce point.

3.2. UNE SEMANTIQUE, POUR QUOI FAIRE ?

3.2.1. Une sémantique qui n'a pas de sens ?

Souvent la sémantique du langage naturel se ramène à l'étude de ce qui relie signifiant (mots, phrases, ...) et signifié (objets, actions, ... du monde réel). On cherche alors une procédure de décision, semblables à des contraintes d'intégrité des bases de données (cf épigraphe), pour savoir si oui ou non une phrase a un sens. Mais est-ce la bonne façon de poser les problèmes ?

Illustrons notre propos par un exemple de Kayser [KAYS 87]. Si l'on fait l'hypothèse, comme cela est souvent le cas dans les systèmes de TLN (et du nôtre en particulier), d'associer à chaque mot un concept défini dans un réseau, comment rendre compte, par exemple, des différents emplois du mot "livre" dans les exemples suivants:

```
"Ce livre se trouve dans toutes les bonnes librairies"
    (élément déterminé ou non d'une classe de livres ?)
"Ce livre a été tiré à 15000 exemplaires"
    (entité à partir de laquelle les éléments de cette classe ont
été créés)
"Jean est parti à la campagne pour écrire un livre"
    (sens figuré du mot "livre")
"Ce livre a fortement influencé les révolutionnaires de 1789"
    (les idées contenues dans ce livre ont eu cet effet)
"Ce livre a été un fiasco pour son éditeur"
    (désignation de toute l'activité sociale liée au livre:
rédaction, lecture, publication, ...)
```

Si la représentation du sens de livre se traduit par un prédicat *livre(X)* qui prend la valeur *vrai* si et seulement si un ensemble de conditions sont satisfaites, nous devrons alors donner plusieurs représentations pour couvrir les usages du mot dans les exemples précédents. Se pose alors le problème complexe de savoir quelle représentation est associée à tel ou tel usage.

Kayser pose les problèmes autrement. Pour lui le sens n'est pas un attribut de la phrase. L'activité de compréhension n'est pas la traduction de la phrase dans une forme canonique, mais est dirigée par des règles de transformations. Le processus de compréhension se fait par approximations successives, chaque tentative déclenchant certaines règles qui permettent de conduire une série d'inférences aboutissant à des niveaux de contradictions plus ou moins approfondis. On a alors des éclairages différents sur les processus de compréhension d'une phrase.

Notre propos n'est pas ici de détailler, cette approche particulière. Nous voulons simplement faire remarquer qu'en abordant la sémantique du langage naturel sous l'angle des rapports signifiant-signifié, en recherchant la bonne forme canonique dont l'interprétation se rapprochera le plus d'un modèle supposé du monde, nous faisons une hypothèse fortement réductrice.

Si nous utilisons des formules logiques pour coder le "sens" d'une phrase c'est parce que ces langages artificiels sont clairs, disposent de bonnes propriétés pour nous permettre de faire des inférences assez simplement et se prêtent bien (au moins pour certaines d'entre elles) à l'implantation. Cette réduction de la sémantique du langage naturel est acceptable dans la mesure où les problèmes que nous avons à traiter sont limités.

Ces choses étant dites, parlons maintenant sémantique et formule logique intermédiaire.

3.2.2. Sémantique et formule logique intermédiaire

Quels types de questions se pose-t-on généralement lorsque l'on cherche les formes canoniques (et éventuellement leurs interprétations) dans lesquelles traduire des phrases:

- quels phénomènes représenter ?

- avec quel langage, quelle sémantique ?
- comment construire cette représentation ?
- cette représentation est-elle complète ?

Détaillons chacun de ces points.

3.2.2.1. Quels phénomènes ?

Parmi les innombrables phénomènes mis en jeu dans la langue (temps, action/état, quantification, pluriels, références, intensionnalités, croyances, présuppositions, ...), nous nous sommes intéressés, au début, à des problèmes de quantification simples, et de références. La quantification est sans doute le problème qui a le plus intéressé (et intéresse encore) le milieu d'informatique linguistique. Ce n'est pourtant qu'un aspect très particulier du langage, et ce phénomène est assez insignifiant dans les textes de spécifications de ACTES[1]. Il est tellement particulier que, parfois, à la lecture des exemples de phrases données dans la littérature du TLN, on a du mal à imaginer un locuteur s'exprimant d'une façon aussi compliquée, et on a plutôt l'impression de voir paraphraser en langage naturel des formules logiques. Ceci dit, la prise en compte de ce phénomène a guidé notre choix du formalisme à associer à notre grammaire syntaxique.

3.2.2.2. Langage intermédiaire et sémantique

Dans le premier chapitre, nous avions qualifié la traduction du langage naturel dans une formule intermédiaire, d'*interprétation sémantique*., suivant en cela la terminologie utilisée par une grande partie de la communauté TLN. Cette *traduction* est qualifiée de "sémantique" car elle permet de révéler les structures caractéristiques des expressions du langage naturel. Nous avions pris soin de distinguer ce terme de la notion d'*interprétation en logique*.

En fait les choses ne sont pas si simples dans les approches du TLN qualifiées de 'sémantique logique' ou 'sémantique formelle'. Derrière ces appellations, nous regroupons les travaux comme ceux de Colmerauer [COLM 79], Pereira [PERE 82], Montague [MONT 74], Barwise [BARW 83] et Kamp [KAMP 81]. Leur but est de caractériser formellement les notions d'énoncé *vrai* et de conséquence dans un modèle (du monde réel ?). Pour cela ils traduisent, en général, les énoncés en langue naturelle dans un langage logique intermédiaire (logique des prédicats du premier ordre standard ou étendue, logique multivaluée, logique intensionnelle, logique situationnelle,...). Cet énoncé en langage logique peut ensuite être interprété dans un modèle (cf figure 3.1).

---

[1] Nous revenons sur les raisons historiques de cet intérêt et de son adéquation aux textes dans la partie suivante en présentant quelques systèmes de traitement automatique de textes.

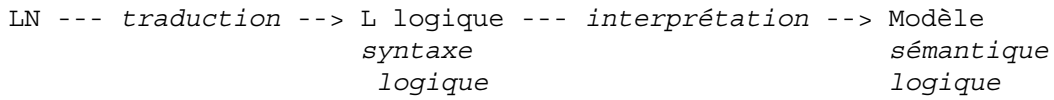


Fig. 3.1: Etapes pour la construction du sens d'un énoncé en 'sémantique formelle'

Qu'est-ce qu'un énoncé vrai dans un modèle ? En logique[2], les choses sont claires. Etant donné un langage L, il suffit de se donner un ensemble d'objets D (*domaine* ou *univers d'interprétation*) et une *fonction d'interprétation* I qui relie constantes du langage L et objets du domaine D et qui, à toute énoncé correcte (bien formé) de L assigne une valeur de vérité *vrai* ou *faux*. La donnée de D et I détermine un modèle M. Une expression de L peut être vraie dans un modèle M1 et fausse dans un modèle M2. D'où les notions de *validité*, *contradiction*, *conséquence logique* pour les expressions de L.

Qu'en est-il en ce qui concerne le langage naturel ? En sémantique formelle, on défend l'idée que les rapports du langage naturel à l'univers extra-linguistique auquel il renvoie peuvent être du même type que les rapports institués, pour une logique , par une interprétation, entre le système formel et l'univers d'interprétation (cf figure 3.2).

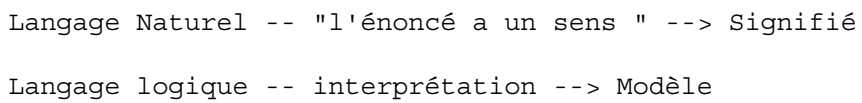


Fig 3.2: Parallélisme entre langage naturel et logique

On a alors deux notions de 'sémantique'. La première, la sémantique logique, permet de définir clairement la façon dont il faut comprendre les énoncés des formules logiques intermédiaires. Elle est importante notamment si les notations utilisées dans ces formules ne reprennent pas les standards de la logique des prédicats. La seconde sémantique est celle qui établit une correspondance entre le langage naturel et le modèle logique censé représenter le monde (cf figure 3.3). Confondre ces deux notions revient à soutenir l'idée que langage naturel et langage logique relèvent des mêmes principes, comme le dit clairement Montague (cf épigraphe)[3], mais n'osent pas toujours le reconnaître les autres sémanticiens du langage naturel.

---

[2] En logique, on distingue les notions de "sens", "signification" et "dénotation" (cf les travaux de Montague pour une discussion technique). Ici nous confondrons "sens" et "signification" car les deux termes relèvent de la sémantique.

[3] Sa Grammaire Universelle sert, d'après lui, à caractériser aussi bien les phrases bien formées du langage naturel que celle de la logique intensionnelle qu'il utilise.

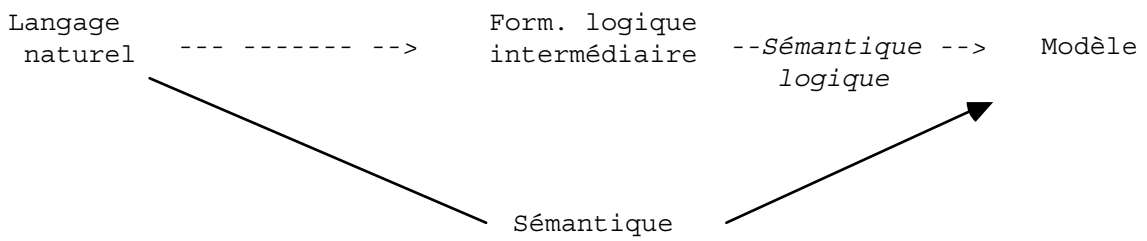


Fig 3.3: Quelle sémantique ?

Les choses ne sont évidemment pas claires sur ce point. On peut noter d'ailleurs, que, bien que les sémanticiens s'intéressent à la sémantique de préférence à la syntaxe, leurs articles portant sur les travaux de Montague décrivent essentiellement la syntaxe utilisée par celui-ci, c'est-à-dire la logique intensionnelle, et rarement la sémantique de cette logique. On peut arguer qu'en parler est chose compliquée compte-tenu de la lourdeur de ses notations. Mais alors cette sémantique a-t-elle un rapport quelconque avec le monde réel ? Nous pouvons parler du monde réel, mais pas de la sémantique censée représenter nos propos !

Pour notre part, nous nous contenterons de dire que nous "faisons de la sémantique" en traduisant le langage naturel dans des formules logiques intermédiaires (nous reprenons donc la définition du premier chapitre). Ces formules seront les structures de représentation de la DRT, les DRSs. Nous utilisons cette théorie (entre autres raisons) parce qu'elle nous offre une façon élégante de relier des formules logiques entre elles lors de l'analyse d'un texte, chose que nous ne pouvons faire en logique des prédicats en restant dans le premier ordre. Mais nous restons malgré tout dans le cadre de cette logique standard (toute DRS peut se réécrire en logique du premier ordre) et pouvons donc utiliser ses propriétés déductives traditionnelles. La sémantique de la DRT n'est pour nous qu'une façon de bien comprendre ce qu'est une DRS bien formée afin de pouvoir lui appliquer des méthodes de preuves.

3.2.2.3. Construire cette représentation intermédiaire

Une représentation sémantique peut se construire de deux façons: compositionnelle (la signification du tout est obtenue à partir de la signification des parties) ou par reconnaissances (tentative de compréhension globale en s'appuyant sur des îlots de "sens" dans la phrase et sur le contexte).

La **construction compositionnelle** est le plus souvent utilisée. C'est un des points forts de la DRT de s'attacher à construire le sens d'une phrase en suivant la façon dont elle est analysée, de gauche à droite[4]. On dit souvent que la construction sémantique se fait parallèlement à celle de la syntaxe (reprenant en cela Montague). Ce qui n'implique pas qu'analyses syntaxique et sémantique doivent être faites simultanément.

Ce dernier problème se pose au niveau de l'efficacité informatique. En phase de test on peut, par exemple, développer une grammaire syntaxique puis une grammaire sémantique qui opérera en entrée à partir de l'arbre syntaxique généré par la précédente grammaire. Cela permet d'isoler les types d'erreurs éventuelles survenues dans l'écriture des règles. Après la mise au point, en phase d'analyse du texte, syntaxe et sémantique peuvent opérer simultanément afin d'appliquer au plus tôt les contraintes sémantiques pour rejeter des phrases incorrectes du texte. On peut également décider de séparer les deux. Cela permet notamment de changer la portée des quantificateurs sans avoir besoin de refaire une analyse syntaxique. Dans notre système, l'utilisateur peut choisir l'une des deux tactiques à sa convenance.

La construction compositionnelle est souvent préférée car elle modélise mieux, dit-on, le processus de compréhension du locuteur/auditeur et qu'elle s'implante plus facilement. Mais elle implique que l'énoncé à analyser soit exempt de toute erreur syntaxique ou sémantique.

La **construction par reconnaissance** , même si elle est plus difficile à implanter, permet de prendre en compte les phrases mal formées. Ce type de phrases est relativement fréquent dans le langage, et n'empêche souvent pas un auditeur humain d'en saisir le sens général à défaut des points de détails. Nous renvoyons au premier chapitre pour les références aux systèmes qui s'appuient sur cette stratégie.

Un bon système de compréhension du langage naturel devrait, sans doute, avoir une stratégie mixte de construction du "sens". Quelques systèmes appliquent une stratégie par reconnaissance lorsque la première analyse a conduit à un échec.

Il existe un autre niveau de problème dans la construction de la représentation sémantique, celui concernant les **possibilités d'implantation** du formalisme. Deux des principales approches en sémantique formelle, la Théorie des Situations de Barwise et la grammaire de Montague utilisent des logiques non standard qui se sont révélées difficiles à mettre en oeuvre informatiquement jusqu'à présent. La Théorie du discours, elle, se prête bien à l'implantation, au moins pour le fragment que nous exposerons. Dans le chapitre suivant, nous présenterons des variantes d'implantation à notre méthode qui est décrite au chapitre 5.

---

[4] Cette affirmation est à nuancer car Kamp n'insiste pas particulièrement sur ce point, au contraire de certains autres partisans de la DRT, dont Klein.

3.2.2.4. Une représentation complète ?

Il nous reste à examiner le problème de la complétude de la représentation sémantique. Prend-elle en compte les connaissances du domaine, le contexte du discours (cf chapitre 1)?

La DRT rend compte dans son algorithme de construction des DRSs d'une partie du **contexte**, en posant le problème du rattachement d'une nouvelle phrase au contenu d'un énoncé précédemment analysé et en caractérisant certains phénomènes d'anaphores. Mais les phénomènes d'action et d'intention ne sont pas traités aujourd'hui par la théorie. En particulier, on ne peut représenter un discours faisant intervenir plusieurs locuteurs. La DRT s'applique donc à un texte et non à un discours au sens où nous l'avions initialement défini. Cela est suffisant pour notre problématique.

Mais la DRT ne s'intéresse pas à la prise en compte des **informations du domaine** ce qui n'en fait pas un vrai langage de représentation des connaissances au sens courant en IA. Kamp la caractérise (cf épigraphe) cependant comme langage de représentation des connaissances parce que son mécanisme de construction du sens est en étroite relation avec la langue. D'autres partisans de cette théorie la jugent apte à représenter tout savoir (cf la discussion au chapitre 4).

Cette mise en correspondance des symboles du langage logique avec des objets du domaine d'un texte ne préoccupe souvent pas beaucoup les sémanticiens du langage naturel. L'analyse de "un livre", par exemple, donnera lieu à la construction d'une formule prédicative *livre(X)* et d'un référent du discours X, sans qu'il soit possible, dans la DRT standard, de connecter ce référent avec un ensemble de connaissances sur ce qu'est un livre.

3.2.2.5. Quelques références

Dans cette introduction, nous ne nous sommes pratiquement intéressés qu'à la DRT. D'autres approches sémantiques logiques ont su bien représenter les problèmes de quantification en logique du premier ordre, notamment dans des systèmes de questions-réponses tels [WOOD 78], [SAIN 86], Colmerauer et Pereira déjà cités. Barwise et Cooper [BARW 81] ont étudié des quantificateurs plus complexes (*la plupart*, *plus de*, *plusieurs*,..) dans une logique différente du premier ordre. Pour une comparaison critique sur ces approches, nous renvoyons à [SEDO 87]. Schubert et Pelletier [SCHU 84] ont complété une grammaire GPSG avec une représentation sémantique prenant en compte des phénomènes d'intensionnalité, tout en restant dans le cadre de la logique standard. Toute ces approches restent dans le cadre de l'analyse de phrases simples.

En ce qui concerne les travaux de Montague, on peut se reporter aux articles de l'auteur déjà cités ou à une présentation exhaustive (sémantique comprise) et critique en français de Chambreuil [CHAM 89]. La sémantique des situations de Barwise s'attache à décrire des aspects du discours non abordés dans la DRT, comme les intentions et modalités. Un article de Cooper [COOP 87] ébauche une comparaison des théories de Montague et Barwise.

Signalons l'approche de Fauconnier [FAUC 84] à la compréhension d'un discours qui aborde, comme la DRT, le problème en termes de construction d'espaces de vérité. Grâce à sa notion d'espaces mentaux, il peut représenter certains phénomènes de références. Nous ne savons pas si ce travail a donné lieu à une implantation informatique.

3.3. LA THEORIE DE LA REPRÉSENTATION DU DISCOURS (DRT)

Nous présentons, dans cette partie, la DRT (Discourse Representation Theory) telle qu'elle a été formulée par Kamp à l'origine. Nous n'abordons pas les questions relatives au temps (imparfait passé simple du français) qui avaient motivé en partie Kamp pour élaborer cette théorie. Dans cette partie, notre présentation de la DRT n'est pas critique. Dans la partie suivante, nous introduisons le problème de la résolution d'anaphores dans les textes et rediscutons alors d'une façon critique de la position de la DRT sur ce problème.

3.3.1. Processus de construction du sens d'un texte

La DRT ([KAMP 81], [KAMP 88]) est une théorie sémantique du discours, c'est-à-dire que l'on peut associer aux structures qu'elle manipule des valeurs de vérité dans un modèle. Cette définition des valeurs de vérité ne dépend pas directement des phrases du texte, mais de la façon dont sont construites les DRSs (Discourse Representation Structure). En particulier, si un discours d est composé des phrases d1,d2, ..., dn , la DRS de d est obtenue par **composition** des DRS de d1,d2,..., dn. La construction de la DRS d'une phrase dépend des DRS produites précédemment.

Cette approche est distincte des traductions sémantiques en logique des prédicats, souvent utilisées en linguistique informatique, où à chaque phrase correspond une formule logique et un discours est la simple **conjonction** de ces formules.

La forme d'une DRS consiste en une paire (Univers,Conditions), où **l'Univers** est un ensemble de **référents du discours** (correspondant chacun à un élément du modèle) et la partie **Conditions** un ensemble de conditions (équivalentes à des formules logiques), représentant les contraintes sur ces référents du discours et ceux définis dans les DRS gouvernant la DRS courante.

L'important ici n'est pas tant le fait que la DRT décrit les conditions pour lesquelles une DRS est vraie dans un modèle mais qu'elle décrit le **processus** permettant de la construire.

Kamp, en 1981, a présenté une grammaire d'un petit fragment de l'anglais et un ensemble de règles de formation des DRS opérant parallèlement à **l'analyse syntaxique** des phrases du texte dérivables de sa grammaire.

La figure 3.4 représente sous forme de boite la DRS associée à la phrase:

```
Un paramètre possède une info de validité.
```

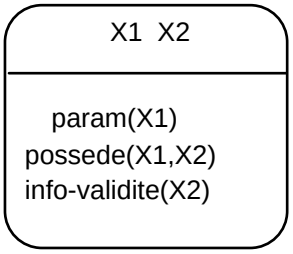


Fig 3.4: DRS correspondante à la phrase ci-dessus.

Cette DRS contient deux référents du discours (Dref, pour *Discourse Referent*, à partir de maintenant) *X1* et *X2*, et un ensemble de conditions sur ces référents. Les conditions de vérité se définissent comme en logique du premier ordre. Cette DRS est vraie dans un modèle M, s'il existe une fonction *f* (dite fonction d'enchâssement[5])qui fasse correspondre à chaque Dref de l'univers de la DRS un élément du domaine de M, telle que chacune des conditions soit vraie dans ce modèle ; c'est-à-dire s'il existe une fonction *f* qui associe à *X1* l'élément *a*, à *X2 b*, tel que *a* est un paramètre, *b* une info de validité et *a* possède *b*.

Grâce à cette notation, Kamp ne traduit pas explicitement les articles indéfinis "un" et "une" par des quantificateurs existentiels (nous verrons l'intérêt de cette démarche un peu plus loin). Ce sont des variables libres dans sa théorie qui prennent une certaine "force" existentielle avec la définition des valeurs de vérité de la DRS, parce qu'il existe une fonction qui introduit les conditions de satisfaction (des conditions de la DRS).

Cette traduction des indéfinis a pour effet de les traiter comme des antécédents potentiels d'un pronom tant qu'il ne sont pas dans la portée d'un universel ou d'une négation. On peut aussi présenter la chose de la façon suivante. Un indéfini a une portée maximale par défaut tant que quelque chose d'autre ne lui impose pas sa propre portée. Lorsque d'autre quantificateurs sont introduits, ils restreignent la portée de l'indéfini à leur propre portée.

---

[5] Cette fonction d'enchassement nous paraît très proche de la fonction d'assignement de la logique des prédicats du premier ordre.

Si, nous continuons le texte avec la phrase "Elle est positionnée à valide", nous obtenons une extension de la première DRS représentée par la figure 3.5. Le pronom introduit un nouveau Dref qui est mis en relation d'égalité avec celui correspondant à "info de validité". La relation anaphorique est donc résolue.

```
Un paramètre possède une info de validité.Elle est positionnée à
valide.
```

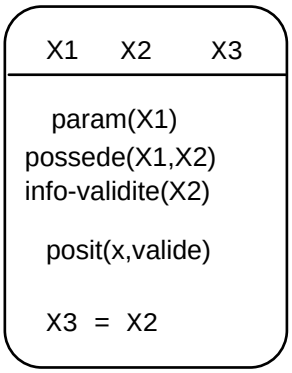


Fig 3.5: Extension de la DRS de la figure 3.4

3.3.2. Universel et négation

Considérons le quantificateur universel et la négation. Ils introduisent des DRS de structures plus complexes.

```
Tout paramètre possède une info de validité.
```

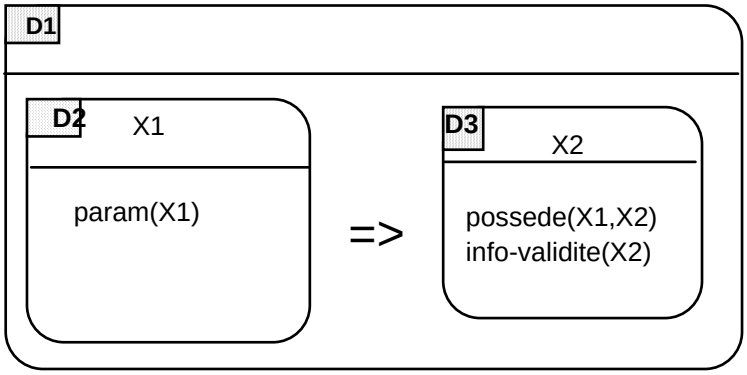


Fig 3.6: DRS conditionnelle associée à un universel

Un paramètre ne possède pas une info de validité.

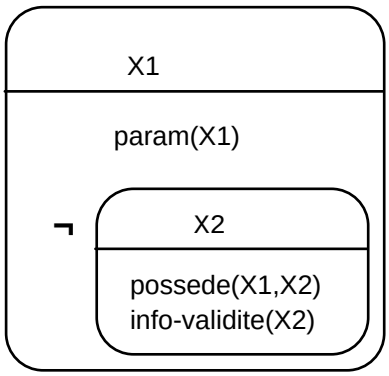


Fig 3.7: DRS associée à une négation[6]

Les deux exemples introduisent des DRS subordonnées dans les DRS principales.La DRS principale de la figure 3.6 contient une condition implicative composé d'une DRS D2 antécédente et D3 conséquente Dans les deux figures le référent du discours *X2* correspondant à "une info de validité" est situé dans une DRS subordonnée. Cela traduit sa portée restreinte dépendante de la portée de l'universel. X2 n'est maintenant plus accessible depuis la DRS principale. Cette traduction restreint les possibilités de référence comme nous le verrons plus loin.

La condition implicative de la DRS de la figure 3.6 est vraie ssi pour toute fonction d'enchâssement *g* qui assigne *X1* à un objet *a* du domaine, *a* étant un paramètre, il existe une fonction *h*, extension de *g*, qui se comporte de la même façon que *g* et en plus assigne *X2* à un objet *b* qui est une info de validité et satisfait la condition *a* possède *b*. La condition de la figure 3.7 est vraie ssi pour toute fonction *g* trouvant un objet paramètre alors il ne peut y avoir d'extension trouvant un objet info de validité tel que le premier objet possède le second.

3.3.3. Conditions d'accessibilité

Ce sont les conditions d'accessibilité entre DRSs et référents du discours qui conditionnent la résolution d'anaphores dans la DRT. L'antécédent d'une anaphore sera recherchée parmi les Drefs accessibles. Voici les définitions de l'accessibilité logique:

[6] La DRS négative comprend les conditions introduites à la fois par le verbe et le groupe nominal. On remarquera la différence avec d'autres traductions logiques plus "classiques" qui feraient porter la négation uniquement sur le prédicat introduit par le verbe.

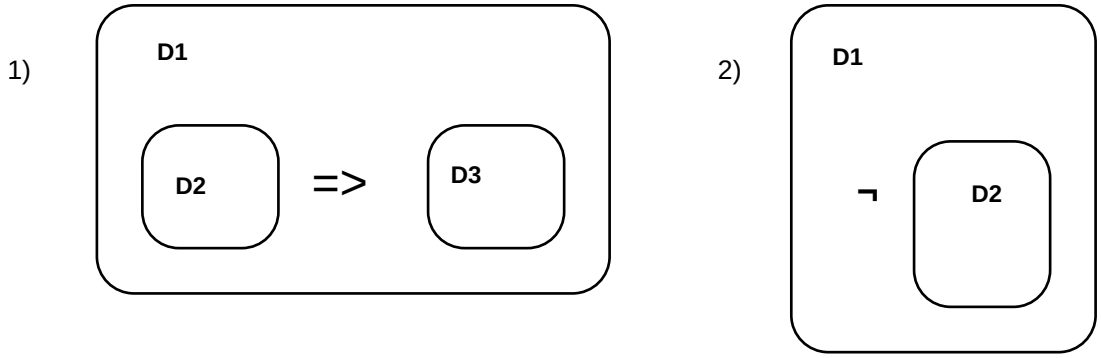


Fig 3.8: Accessibilité entre DRSs

Accessibilités pour les DRSs:

- toute DRS est accessible à elle-même.
- en 1) et 2), D1 est accessible de D2.
- en 1), D2 est accessible de D3.
- l'accessibilité est transitive: si D1 est accessible de D2, de D2 est accessible de D3 alors D1 est accessible de D3.

Accessibilité pour les Drefs:

Un référent du discours X1 est accessible d'un autre référent X2, si la DRS minimale contenant X1 est accessible de la DRS minimale contenant X2.

3.3.4. Un exemple de type "donkey sentence"

Geach a proposé en 1962 des types de phrases (baptisées "donkey sentences" parce que parlant de fermiers - *farmers*- et d'ânes- *donkeys*- battus) contenant des anaphores insolubles si on prenait le parti de traduire ces phrases en logique des prédicats standard de façon compositionnelle. En voici un exemple adapté à notre domaine:

```
Tout paramètre qui possède une info de validité est conditionné par
elle.
```

La coréférence est problématique si l'on traduit cette phrase de gauche à droite en associant à "un" un quantificateur existentiel. Le symbole correspondant à "elle" n'est pas dans la portée de l'existentiel. L'anaphore est insoluble.:

$\forall p \{ (param(p) \wedge \exists i [info\text{-}val(i) \wedge possede(p,i)]) \rightarrow conditionner(?\mathbf{elle}, p)\}.$

Voici la traduction qu'en donnerait Kamp (figure 3.9):

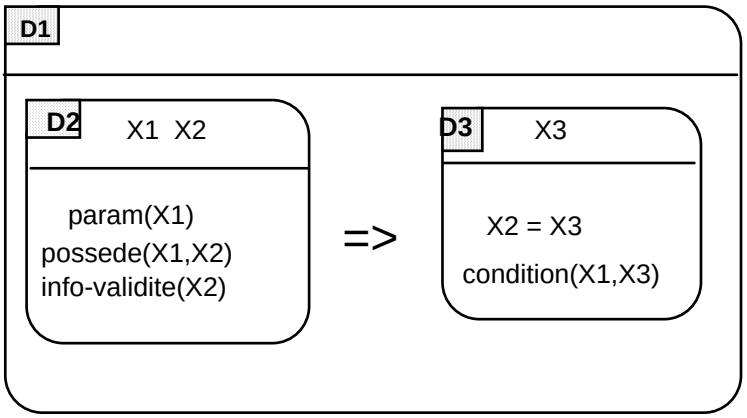


Fig. 3.9 : Traduction d'une "donkey sentence".

Le déterminant "tout" introduit le référent du discours X1 et une condition implicative. La relative est traduite dans la DRS antécédente D2. La coréférence est explicite entre "une info de validité" et "elle" car le marqueur X2 de la DRS D2 est accessible au marqueur X3 de D3.

C'est probablement ces types de phrases qui ont amené Kamp à ne pas introduire de quantification explicite pour les indéfinis et à leur donner par défaut une portée maximum (comme l'a fait également Heim [HEIM 82]).

En effet, en logique des prédicats standard, dans une implicative, si une variable ne figure pas dans la partie droite, il y a équivalence entre quantifier existentiellement cette variable sur la partie gauche seulement et la quantifier universellement sur toute l'implication. Mais seule la quantification universelle permet de faire apparaître également la variable en partie droite. Autrement dit le quantificateur existentiel limite la possibilité d'extension de la formule avec coréférence à cette variable à la partie gauche, tandis que l'universel l'autorise en partie gauche et droite.

$(\forall x\ [\ P(x) \rightarrow Q]) \leftrightarrow ([\ \exists x\ P(x)] \rightarrow Q)$ avec Q ne dépend pas de x

L'idée de Kamp a consisté, à partir de la constatation de l'équivalence partielle entre existentiel et universel, à ne plus considérer (implicitement) que des existentiels et des structures conditionnelles. Ce remplacement existentiel/universel par existentiel/conditionnelle rend donc la formule logique un peu plus compositionnelle.

3.3.5. Continuation du discours/texte

Illustrons sur des exemples le fait que la définition des conditions d'accessibilité et les traductions des universels et des négations bloquent certaines références.

La phrase de l'exemple 3.9 ne peut être continuée par "Si elle est à invalide". En effet, *elle* ne peut référer à 'info de validité' puisque le marqueur X2 n'est pas accessible à X4 (cf figure 3.10).

Tout paramètre qui possède une info de validité est conditionné par elle. Si *elle* est à invalide ...

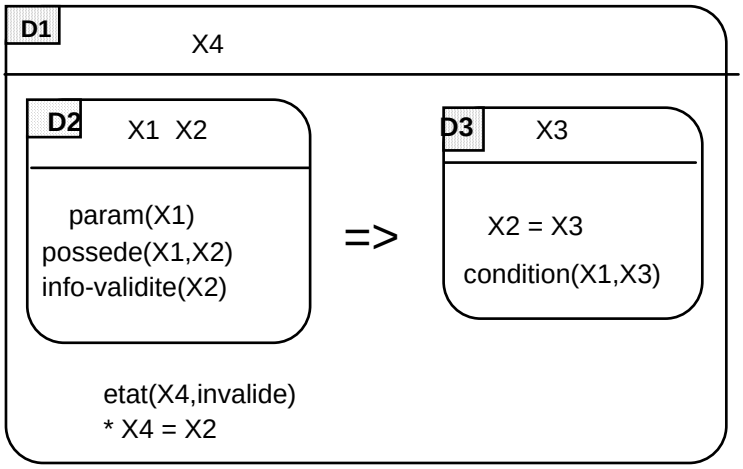


Fig. 3.10 : Contraintes sur la continuation du discours après un universel.

La même remarque peut s'appliquer après une négative (cf figure 3.11).

Un paramètre ne possède pas d'info de validité. Si *elle* est à invalide...

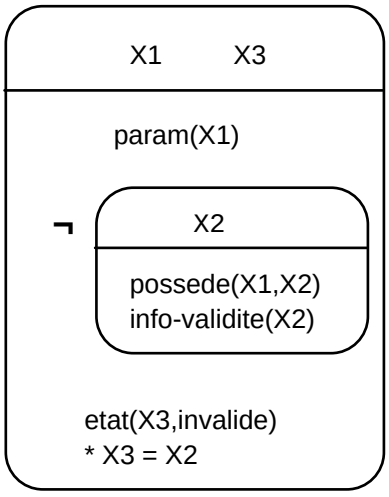


Fig. 3.11:Contraintes sur la continuation du discours après une négation.

Ainsi donc, ce qui importe pour le traitement des anaphores dans la DRT, c'est l'accessibilité de la position du référent du discours, pas les conditions dans lesquelles figurent ce marqueur.

Les Drefs correspondants aux **noms propres** et à quelques **descriptions définies** sont systématiquement placés dans la DRS principale du texte, quel que soit l'endroit du texte où les mots ont été mentionnés. Ces Drefs sont donc toujours accessibles. Dans l'exemple de la figure 3.12 nous traitons "vitesse-conventionnelle" comme un nom propre parce que c'est une information unique bien déterminée, identifiable par ce nom. L'anaphore est acceptable, puisque X2 est accessible. Les conditions se rapportant à ce référent du discours sont dans une DRS subordonnée.

```
Un paramètre ne dépend pas de la vitesse-conventionnelle. Si elle
est à invalide ...
```

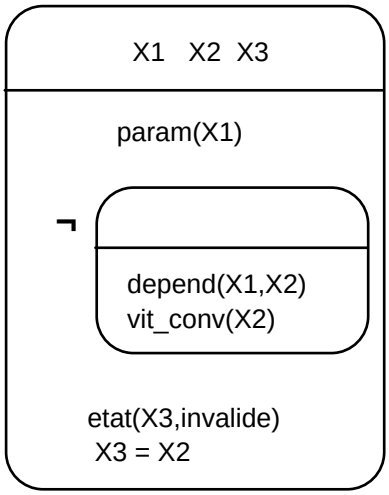


Fig. 3.12: Le référent correspondant à un nom propre (ou l'équivalent d'un nom propre) est toujours accessible

3.4. LA RESOLUTION D'ANAPHORES DANS LES TEXTES

Après avoir présenté la DRT, nous allons maintenant parler plus précisément des anaphores dans les textes, en définissant cette notion de façon plus précise, en évoquant les mécanismes complexes mis en jeu dans leur résolution. A la lumière de cette présentation nous critiquerons les affirmations de la DRT sur les références anaphoriques[7].

[7] Nous ne faisons pas directement référence aux positions de Kamp énoncées dans les articles de 1981 et 1988, car celui-ci ne s'interesse pas directement au problème de la résolution des anaphores mais vise à offir une théorie 'explicative' de l'anaphore. Les positions que nous critiquerons sont celles prises par les défenseurs de la DRT qui ont appliqué cette théorie à la résolution des anaphores(cf les citations dans le texte). Nous pensons que leurs positions reflètent bien la pensée de la majorité des chercheurs travaillant sur la DRT. Elles n'ont pas, à notre connaissance, été critiquées.par d'autres partisans de la DRT.

3.4.1. Le problème de l'anaphore dans les textes

La référence est omniprésente dans les textes en langage naturel et les dialogues hommes-machines. La co-référence est l'un des éléments essentiels assurant la cohésion. Les systèmes de compréhension de textes ou de gestion du dialogue accordent donc une place de choix à la résolution d'anaphores, mais ils n'implantent souvent que des méthodes rudimentaires et partielles. Cela tient à la complexité du problème.

Il existe en effet ***plusieurs types d'anaphores*** : défini, non-défini, pronom, référents singuliers, pluriels,... . Les ***sources de contraintes*** (permettant d'accéder à des antécédents potentiels d'un terme référentiel, de les choisir ou rejeter) sont multiples: genre et nombre d'un antécédent, focus local et global du discours, connaissances du monde, pragmatique du discours, accessibilité logique,.... Les ***sources de connaissances*** nécessaires à l'expression de ces contraintes ont des représentations variées : arbre syntaxique, représentation sémantique intermédiaire, règles pragmatiques. Enfin l'***accès*** à ces connaissances et l'***évaluation des contraintes*** correspondantes peuvent intervenir à des moments différents du processus d'analyse d'un texte : pendant l'analyse d'une phrase, à la fin de l'analyse d'une phrase, après le traitement d'un paragraphe ou d'une partie.

3.4.2. Anaphores et références

L'anaphore est un cas particulier de relation entre deux expressions du langage assurant la cohésion d'un discours écrit ou parlé [CART 87]. Dans une relation anaphorique le sens d'un des éléments (le **terme anaphorique**), pris isolément, est vague ou incomplet et peut seulement être compris correctement en considérant le sens de l'autre élément (l'**antécédent**) avec lequel il est en relation. Ainsi dans la phrase :

> Les citrons verts sont plus fins et *ils* ont meilleur goût que les jaunes.

le groupe nominal "les citrons verts" introduit les citrons verts dans l'univers du discours. Le pronom "ils" est en **relation anaphorique** avec "les citrons verts" car il fait **référence** à ce même objet du monde mentionné dans le discours. C'est cette co-référence qui nous permet de comprendre le sens du mot "il" et non la simple relation entre des mots de la langue[8].

Sans cette référence aux objets de l'univers du discours, on ne pourrait expliquer l'anaphore suivante:

---

[8] Dans cette partie nous suivons la terminologie introduite par Carter et donc n'utiliserons pas le terme "référence" pour caractériser la relation entre deux expressions de la langue, comme le font parfois certains linguistes. On notera de plus que notre notion de l'anaphore est très générale. Elle ne se limite pas aux cas des anaphores pronominales.

Je viens de manger un citron vert. *Ils* sont vraiment très fins.

"Ils" fait référence à l'ensemble générique des citrons verts. L'anaphore est explicite si l'on admet que "un citron vert" réfère à la classe des citrons verts.

Mais lors du processus d'analyse d'un texte, on n'accède pas directement aux éléments d'un monde (réel ou imaginaire) décrit par le texte. On construit peu à peu un modèle partiel du contenu du texte qui sera, dans une phase ultérieure, interprété dans ce monde. Ainsi l'analyse du groupe nominal "un citron vert" **spécifie** l'élément du modèle *Citron_vert_1* qui lui même **représente** un citron vert (Fig. 1).

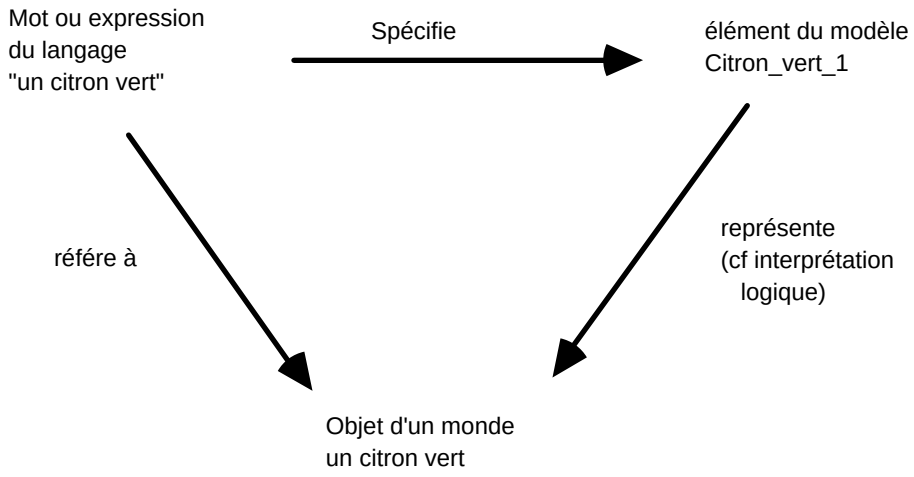


Fig. 3.13 : Référence, spécification et représentation.

Nous utiliserons généralement le terme **référent du discours** ([KAMP 81], [KART 76]) pour désigner l'entité introduite par un groupe de mots (essentiellement un groupe nominal) dans le modèle partiel du discours. Le terme "d'élément cognitif" est également utilisé par [SIDN 83]. La **résolution d'une anaphore** consiste donc à établir une relation entre deux référents du discours (celui introduit par le terme anaphorique et celui déjà présent dans le modèle du discours, l'antécédent) et vérifier qu'ils désignent bien le même objet du monde.

3.4.3. Stratégies, connaissances multiples

Voici quelques exemples (non exhaustifs) d'anaphores montrant la diversité des connaissances et des stratégies impliquées dans la recherche d'antécédents (pour une présentation exhaustive voir [HIRS 81] et [REIN 83]).

Genre, nombre, type

Certains traits syntaxico-sémantiques du référent anaphorique imposent des contraintes sur la recherche d'antécédents. Ainsi, le genre permet de rejeter toute coréférence entre 'Paul' et 'elle' dans l'exemple:

```
Paul et Isabelle vont à l'école. Elle a un cartable à
pois.
```

Genre, nombre, type,... sont des contraintes locales (connaissances liées aux référents du discours) dont l'évaluation permet de filtrer les antécédents, après la phase d'analyse syntaxico-sémantique.

Référence disjointe :

Il existe un certain nombre de cas bien définis interdisant deux termes référentiels de référer au même objet. Ainsi dans

```
* Lucien¡ le ¡ voit dans la glace.
```

Lucien ne peut être l'antécédent de *le* car l'antécédent d'un pronom personnel ne peut jamais être dans la même clause au même niveau dans la phrase (sauf pour les possessifs et les réfléchis). Guenthner [GUEN 83] décrit un certain nombre de règles sur la référence disjointe. Si ces règles (source de connaissances) sont implantées dans un module de résolution de référents, elles doivent être évaluées au moment de l'analyse syntaxique. Le module de résolution se contentera de rejeter la coréférence entre deux référents du discours disjoints. Un module gérant la référence disjointe peut opérer non seulement comme filtre, mais peut également proposer des candidats à un référent comme dans : "Lucien¡ *se* ¡ voit dans la glace". Dans ce cas les contraintes syntaxiques imposent Lucien comme antécédent.

Contraintes de Préconditons/Postconditions

```
Pierre donne une pomme à Loïc. Il la mange.
```

Pour savoir que le pronom 'il' ne peut faire référence à 'Pierre', le système doit posséder un certain nombre de règles de connaissances sur le monde ([CARB 88]) du genre: un acteur ne possède plus un objet qu'il vient de donner (postcondition) ; pour manger quelque chose, un acteur doit d'abord avoir cette chose (précondition). Il doit après l'analyse du texte accomplir un raisonnement pour trouver qu'il y a contradiction entre préconditions et postconditions.

Règles pragmatiques

Des règles pragmatiques peuvent indiquer suivant le contexte d'analyse les endroits du texte où il faut chercher des antécédents, indiquer les référents du discours qui sont l'objet d'attention principale tout au long (ou pour une partie) du texte. Ces référents font alors partie du focus global. Des exemples en seront donnés plus loin. On notera simplement ici que ces règles ne sont activables que lors de l'analyse de certaines parties du texte.

Ensembles

Certaines stratégies de recherche ne sont activables qu'après une première phase de recherche infructueuse d'un antécédent. Ainsi pour établir la référence entre 'ils' d'une part, 'Pierre' et 'Paul' d'autre part, dans la phrase :

```
Pierre est chez lui, Paul au bureau. Ils doivent se
retrouver ce soir au cinéma.
```

le système pourrait appliquer la stratégie suivante :si le référent est pluriel et qu'aucun antécédent n'a été retenu, essayer à partir de référents individuels de reconstituer un ensemble s'accordant avec la description du référent anaphorique.

3.4.4. DRT et anaphores: retour critique

Quelle peut être la place de la DRT dans un système de résolution d'anaphores ? Dans ACTES, le texte est traduit sous forme de DRSs. Les antécédents potentiels d'une relation anaphorique sont donc déterminés: ce sont les référents du discours des univers des DRSs. On peut utiliser les conditions d'accessibilités sur les Drefs et les règles sur la continuation du discours comme un filtre opérant après l'analyse sémantique. Mais ce filtre ne fait que traduire des contraintes sur la portée des quantificateurs. Deux types de questions se posent alors:

- ce filtre, en relation direct avec la structure sémantique du texte, est-il le plus important ? Doit-il définir le cadre rigide dans lequel s'appliqueront les autres sources de contraintes ?
- ce filtre est-il pertinent pour traduire les contraintes linguistiques dans des phrases comportant des quantificateurs ?

Intéressons nous, pour l'instant, à la deuxième question. Les règles sur la continuation du discours et les conditions d'accessibilité logique sont-elles pertinentes pour contraindre les relations anaphoriques entre phrases quantifiées ?

Les règles sur la continuation du discours en question

La règle de continuation après une phrase "dominée" par un quantificateur universel, présentée en 3.3.5 n'est pas très convaincante. L'anaphore de l'exemple de la figure 3.10 nous semble parfaitement acceptable. Prenons un autre exemple du même genre.

```
(1) Chaque fermier en Corée stocke sa récolte dans une charrue en
bois.
(2) Il attend plusieurs mois pour la vendre.
(3) Elle ne sera vendue qu'après plusieurs mois.
```

Dans la DRT, aucune des phrases (2) et (3) ne peut continuer le texte commencé avec (1), car les pronoms personnels n'ont pas d'antécédents: les Drefs correspondants à "fermier" et "récolte" sont dans des DRSs subordonnées à la DRS principale du texte, donc inaccessibles. La co-référence entre (1) et (3), par exemple, est interdite parce que l'existentiel correspondant "récolte" est dans la portée d'un universel, et deviendrait de ce fait inaccessible dans les phrases suivantes.

Par contre la DRT accepte parfaitement l'anaphore suivante:

```
(4) Chaque fermier en Corée stocke, dans une charrue en bois, sa
récolte qui ne sera vendue qu'après plusieurs mois.
```

Quelle différence entre (4), d'une part, (1) et (3), d'autre part, sinon la formulation quelque peu lourde de (4) ? La règle de bonne formation d'une DRS conditionnelle stipule qu'à la fin de l'analyse de la phrase, la DRS conséquente "se termine" (ou "est fermée") et que le discours doit se continuer dans la DRS principale. L'appliquer telle qu'elle revient à dire qu'un locuteur doit décrire un objet ou une situation en une seule phrase, sans avoir la possibilité de compléter cette description dans les phrases suivantes ! A quoi sert alors l'anaphore ?

Cette formulation est manifestement fausse et ne traduit pas les critères linguistiques qui permettent de décider comment rattacher la DRS partielle d'une nouvelle phrase aux DRS déjà existantes. Dans quels cas faut-il continuer l'analyse dans la dernière DRS conséquente ou bien dans la DRS principale ? La réponse à cette question est importante puisqu'elle permet de structurer correctement tout le texte.

Ces critères gouvernent la résolution des anaphores. Mais avant d'essayer de les déterminer, il serait bon de savoir si nous ne faisons pas fausse route en parlant des relations entre portée des quantificateurs et anaphores.

Portée des quantificateurs et anaphores

L'exemple suivant montre que la nature des relations anaphoriques dépend de la portée des quantificateurs.

(5) Claire a montré à chaque garçon une rose.

(6) Cette rose est d'une nouvelle variété de couleur très proche du noir.

(7) Ensuite elle les a ramassées pour en faire un bouquet.

Dans l'exemple (5) l'interprétation de "une" est ambiguë: est-ce la même fleur qui est montrée aux garçons, où est-ce à chaque fois une fleur différente ? Suivant la réponse, la portée de l'existentiel traduisant "une" dominera ou non l'universel correspondant à "chaque". Si le contexte n'est pas mieux précisé avant (5), il faut attendre d'avoir résolu les relations anaphoriques pour savoir quelle est la bonne interprétation (et non l'inverse): la description définie "cette rose" correspond à une lecture existentielle (la même rose) de "une rose" dans (6) ; le pluriel dans (7), une lecture distributive (une rose par garçon).

On notera que le pluriel n'est pas forcément attaché à une lecture distributive du quantificateur. Le terme anaphorique peut aussi bien être singulier.

(8) Chaque pensionnaire est suivie par une psychologue.

(9) Elle est disponible 24 heures sur 24.

(10) Elles se sont formées sur le tas aux conditions particulières de l'institut.

En (9) on reparle de la psychologue attachée à un pensionnaire comme d'un modèle dont on reprécise l'un des attributs obligatoires. En (10), on s'attache à décrire l'une des particularités du groupe des psychologues (puisque l'on sait qu'il y en a plusieurs).

Après le constat de l'existence d'une relation entre anaphores et quantification, revenons sur la DRT pour savoir si elle nous donne quelques indications sur les critères linguistiques qui gouvernent la résolution des anaphores.

Quels critères linguistiques dans la DRT ?

Cette théorie semble assez bien traduire les contraintes après les **phrases négatives**. Reprenons un exemple de Reyle (cité dans [BLOC 87]), partisan de la DRT:

(11) Susan does not own a book. * She reads it.
(Susan ne possède pas de livre.* Elle le lit)

L'anaphore entre "it" et "a book" est impossible du fait de l'inaccessibilité du Dref correspondant à "a book" (cf figure 3.11). Ce que la DRT traduit ainsi c'est donc le fait qu'il est difficile de reparler d'une situation niée dans le but de compléter sa description (ce cas diffère donc des quantificateurs universels).

Mais l'exemple ne traduit pas que cela. Le rattachement entre "it" et "a book" est d'autant plus impossible (ou difficile à saisir pour un auditeur) qu'il y a glissement de thème entre la première et la deuxième phrase: on parle de possession dans un cas, de lecture dans l'autre. Or la proximité ou l'éloignement sémantique entre deux termes ou deux groupes de mots influent fortement sur le processus de résolution d'anaphores [CARB 88]. Changeons un peu l'exemple:

(12) Susan ne possède pas de voiture.
(13) Elle la loue.
(14) Elle en loue une.

Cette fois le thème de (12) est beaucoup plus proche de celui de (13) et (14). C'est pourquoi, même si (14) est seule grammaticalement acceptable, (13) peut être employée dans une expression familière et sera sans doute facilement interprétée par un humain. La DRT n'accepte pas ces "nuances". Mais, à supposer qu'un analyseur robuste accepte (13), il devrait l'interpréter comme un cas de "une-anaphore" (*one-anaphora)*. La relation anaphorique est dans ce cas essentiellement grammaticale: trouver le constituant manquant après "une". Il ne saurait être question de dire que les référents du discours renvoient au même objet du domaine.

Après les phrases négatives, revenons maintenant aux problèmes des phrases **quantifiées universellement**. Nous avons déjà constaté que la DRT ne nous apporte pas grand chose sur la nature des relations anaphoriques dans ce type de phrases. Souvent d'autres phénomènes sont bien plus significatifs que celui de la portée des quantificateurs. Considérons l'exemple suivant:

(15) Chaque fermier en Corée stocke sa récolte dans une charrue en bois.
(16) Il la protège ainsi de l'humidité.
(17) Elle est souvent en mauvais état et vermoulue.

Si (16) ne pose pas de difficultés de compréhension, (17) est plus problématique, non à cause d'un quelconque problème de portée des quantificateurs mais parce que le foyer d'attention, le focus local, du discours a beaucoup changé entre (15) et (17). Sidner [SIDN 83], Grosz [GROSZ 77], entre autres, ont décrit les règles de gestion du focus dans un texte. Une entité introduite par un groupe nominal en position sujet ou complément d'objet direct est un bien meilleur candidat potentiel d'une relation anaphorique qu'une entité introduit dans une locution de lieu. Pour comprendre (17) le locuteur doit "réaliser" le glissement important de focus.

Un tel exemple montre bien qu'un filtre traduisant (même correctement) les contraintes d'accessibilités logiques entre référents du discours ne saurait représenter seuls les contraintes anaphoriques dans des phrases quantifiées.

3.5. CONCLUSIONS SUR NOTRE APPROCHE DE LA DRT

La DRT nous permet d'obtenir une représentation sémantique intermédiaire structurée d'un texte. Elle offre une façon élégante et facilement informatisable de construire cette représentation en tenant compte du contexte. Les phénomènes pris en compte sont limités et il nous faudra donc l'adapter à nos besoins.

Le fragment d'anglais présenté par Kamp inclut les déterminants indéfinis "a", "every", les conditionnelles (du type "si ... alors"), les noms propres, les relatives, les pronoms personnels "he", "she", "it". Depuis 1981, des extensions ont été proposées, telles le traitement des ellipses de groupes verbaux [KLEI 84]. Aujourd'hui une partie des recherches s'orientent vers la quantification plurielle (première approche dans [EIJC 84]), et la référence temporelle.

Certaines critiques sont adressées à la DRT soutenant qu'elle n'apporte rien de nouveau qui ne puisse être décrit dans d'autres théories. Cette façon de poser les problèmes rappelle la controverse réseaux sémantiques et logique en IA. Si les chercheurs en informatique linguistique se sentent plus à l'aise dans le cadre de la DRT pour formuler leurs problèmes et implantent plus facilement leurs résultats, cela suffit largement à justifier l'intérêt qu'elle suscite. L'important est surtout de pouvoir juger la qualité des résultats en TLN obtenus à partir de cette théorie. Avant de pouvoir répondre sur ce point, il faudra encore attendre.

Ce que nous avons mis en évidence est le peu de pertinence linguistique de cette théorie dans son approche du traitement des anaphores (sur ce point, voir aussi [SELL 85a], [BLOC 87]). Mais il n'existe pas aujourd'hui de théorie unique et complète de résolution d'anaphores. Nous montrerons dans la seconde partie qu'il est possible de construire un système implantant plusieurs théories partielles en les faisant coopérer. Les contraintes d'accessibilités logiques définies par la DRT ne sont alors qu'un des paramètres pris en compte dont les affirmations peuvent éventuellement être remis en cause.

Il existe peu d'autres approches sémantiques dépendantes du contexte: signalons, sur ce point, les travaux de [WEBB 78] et [WEBB 88] qui apportent une contribution importante au problème de la résolution des anaphores.

4. PRESENTATION DE QUELQUES SYSTÈMES

"When considering NL from a systems point of view, issues emerge that can usually be ignored when focusing on particular language phenomena and individual components."

"Bringing principled theoretical approaches to bear effectively on practical, and hence usually idiosyncratic, applications remains a challenge"

"... Although the systems described above were all important efforts, it has been recognized that they were all limited - and limited in more than one respect. That they could be as successful as they were is primarily because they were able to take advantage of their well-defined micro application. In this respect, NL interfaces have generally been more tractable than text processing systems."

Présentation des éditeurs sur les systèmes de TLN dans *Readings in Natural Language Processing*, 1986.

4.1. INTRODUCTION

Dans ce chapitre nous présentons des systèmes de TLN qui se situent dans une approche voisine de la nôtre, c'est-à-dire des systèmes qui ont été construits à partir d'hypothèses de travail voisines de celles décrites dans la présentation. Nous évoquerons d'abord quelques grammaires d'unification qui ont choisi la DRT comme représentation sémantique ; puis des environnements de développement de grammaires d'unification ; et enfin quelques systèmes de compréhension de textes construits à partir de grammaires d'unification et/ou utilisant la DRT.

L'intérêt d'une telle présentation est de faire un tour d'horizon original des systèmes implantant grammaires d'unification et DRT, de savoir si des traits les distinguent d'autres approches plus conventionnelles en TLN, de dégager quelques critères de comparaison permettant de mieux situer ensuite notre travail.

Ne sont pas évoqués dans cette partie les systèmes construits à partir de grammaires logiques (cf la thèse de Sabatier [SABAT 87] sur ce point), ni les systèmes de génération utilisant une grammaire d'unification (cf [KEOW 87] et [SIMO 85]). Les systèmes de résolution d'anaphores sont décrits dans le chapitre 7.

4.2. GRAMMAIRE D'UNIFICATION ET DRT

Voici quelques systèmes implantant une grammaire d'unification et une partie de la DRT. Nous donnons quelques indications sur la grammaire, les structures de traits et la façon d'implanter la DRT et, éventuellement, les a priori de leurs auteurs.

4.2.1. BUILDRS

BUILDRS [WADA 86] est une implantation Prolog d'une partie de la DRT associée à une grammaire LFG, développée par Wada et Asher à l'université d'Austin (où Kamp a travaillé plusieurs années).

BUILDRS est composé de trois modules: un analyseur LFG, un constructeur de DRS, et un module de résolution d'anaphores. Il analyse un texte composé de quelques phrases présentant surtout des problèmes liés aux quantificateurs universels et existentiels. Ses phrases sont traduites en DRS sur lesquelles opère le module de résolution d'anaphores. Les textes sont de caractère "généraux", c'est-à-dire ne faisant intervenir aucune connaissance d'un domaine particulier.

Prenons un exemple pour montrer la façon dont sont construites les DRS. Soit la phrase à analyser: "chaque homme aime une femme". L'analyseur LFG produit la f-structure (structure de traits) suivante:

```
[ SUJ  = [ SPEC = 'chaque' ]
         [ GEN = masc      ]
         [ PRED = 'homme'  ]
         [ NUM = sg        ]

  OBJ  = [ SPEC = 'une'    ]
         [ GEN = fem       ]
         [ NUM = sg        ]
         [ PRED = 'femme'  ]

  PRED =    'aime'<(SUJ)(OBJ)>' ]
```

Fig 4.1: f-structure de "chaque homme aime une femme"

Le constructeur de DRS opère à partir de cette f-structure et de formules de lambda-calcul associées à chaque entrée lexicale. Les formules associées aux déterminants sont des DRS partielles:

```
(1) chaque : < x, λPλQ, ([] => ( ([x],[P]), ([], [Q]) ) >
(2) une : < x, λPλQ, ([x], [P,Q]) >
x est le référent du discours
P et Q représentent des ensembles de conditions.
```

Celles associées à un nom et au prédicat principal de la f-structure sont respectivement:

```
(3) homme: < λx, O, [homme(x)] >
(4) aime: < O,O, [aime(X,Y)] >
X et Y seront unifiés avec les référents du discours introduits par
les spécificateurs SUJ et OBJ
```

L'algorithme construit d'abord la formule correspondante à "chaque homme", puis "une femme", puis "aime une femme" et enfin celle correspondante à la phrase entière:

```
(5) <O,O, ([], => ( ([u1],[homme(u1)]),
([u2],[femme(u2),aime(u1,u2)]) ) ) >
```

L'algorithme peut générer, par backtracks successifs (après échec dans la recherche d'un référent à une anaphore), les autres lectures possibles des portées des quantificateurs. La formule (5) correspond à une lecture distributive (une femme différente par homme). L'autre possibilité (une seule femme aimée par tous les hommes) est générée en construisant d'abord la formule se rapportant à "chaque homme aime" puis en appliquant celle-ci à la formule liée à "une femme", ce qui donne:

```
(6) <O,O, ([u2],[femme(u2), => ( ([u1],[homme(u1)]),
([],[aime(u1,u2)])) ]) >
```

Une telle lecture existentielle, est par exemple provoquée par l'analyse de la deuxième phrase du texte suivant :

"Chaque homme aime une femme. Elle est belle"

La première lecture est distributive. Puis ne trouvant pas d'antécédent pour "elle", le constructeur de DRS propose une lecture existentielle. A aucun moment, n'est envisagée la possibilité de traiter la deuxième phrase comme une continuation de la première et d'étendre en conséquence la dernière DRS produite.

Le module de résolution d'anaphores de BUILDRS est évoquée dans le chapitre 7.

Wada et Asher justifient le choix de la DRT comme représentation sémantique, par le traitement qu'elle propose des anaphores et des groupes nominaux définis et indéfinis. Ils insistent sur la séparation entre deux composants de la DRT: le mécanisme de production des DRS à partir d'un discours, d'une part, l'interprétation de ces DRS dans un modèle, d'autre part. Pour eux, les recherches sur ces points peuvent être menés distinctement. En somme, la résolution d'anaphores ne se préoccuperait pas de savoir à quels éléments du monde réfèrent les référents du discours. Ils reconnaissent que des éléments de connaissance sur le monde sont parfois nécessaires pour aider à résoudre certaines anaphores ... mais ces éléments doivent être introduits avec mesure car ils sont coûteux à manipuler.... alors où sont les problèmes ?

4.2.2. La grammaire d'unification catégorielle (CUG)

Dans le cadre du projet IBM allemand LILOG, Uszkoreit et son équipe ont développé un formalisme de description de graphes utilisant l'unification, formalisme appelé STUF [USZK 88]. Dans STUF, les règles de grammaire sont aussi décrites sous formes de graphes. Toutefois les graphes qui seront utilisés dans la grammaire doivent d'abord avoir été déclarés complètement dans un module approprié.

Uszkoreit s'est servi de STUF pour implanter une grammaire de type catégorielle [USZK 86]. L'avantage d'un tel choix est de réduire considérablement le nombre de règles de grammaires et regrouper toutes les informations dans le lexique. La structuration du lexique et l'utilisation de gabarits, par exemple, prend alors toute son importance. La sémantique en TLN est souvent construite à partir de formules élémentaires de type foncteur-arguments. Il est en de même dans les grammaires catégorielles pour la syntaxe. Un déterminant, par exemple, est un foncteur (noté GN/N) qui prend un nom comme argument et produit un GN comme valeur. Dans ce cas, l'argument doit suivre le foncteur. Un verbe intransitif est un foncteur (PH\GN) qui prend un groupe nominal GN comme argument et produit une phrase PH. Ici l'argument précède le foncteur.

La grammaire se construit donc sur deux règles de réduction essentielles:

```
Application droite : A -> A/B B
Application gauche: A -> B A\B
```

La figure 4.2 montre un exemple d'application de ces deux règles pour analyser une phrase simple.

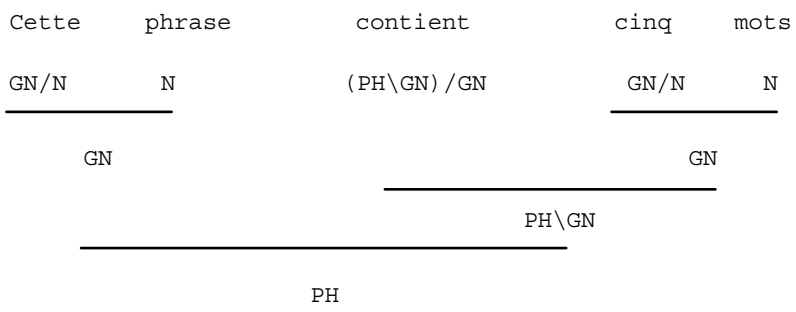


Fig 4.2: Analyse d'une phrase simple avec les deux règles d'application

Ces règles se représentent facilement comme un graphe en STUF La figure 4.3 donne une représentation en STUF et la correspondance sous forme de matrice de la règle d'application droite.

```
appli_gauche : valeur -> foncteur argument
          <foncteur syn val > = < valeur>
          <foncteur syn dir > = gauche
          < foncteur syn arg> = < argument>
```

```
( graphe = [ valeur = <1>
             foncteur = [ syn [ val = <1>
                                dir = gauche
                                arg = <2> ] ]
             argument = <2> ]
)
```

Fig 4.3: Représentation matricielle de la règle *appli_gauche*

La figure 4.4 donne un exemple d'instanciation de la partie *foncteur* de cette règle pour un verbe intransitif. Les traits supplémentaires permettent de construire une formule sémantique prédicative (équivalente ici à *dort(X)*) en unifiant l'argument du prédicat du verbe avec celui venant du groupe nominal.

```
<syn arg sem> = < sem formule arg>
< syn val sem > = < sem >
```

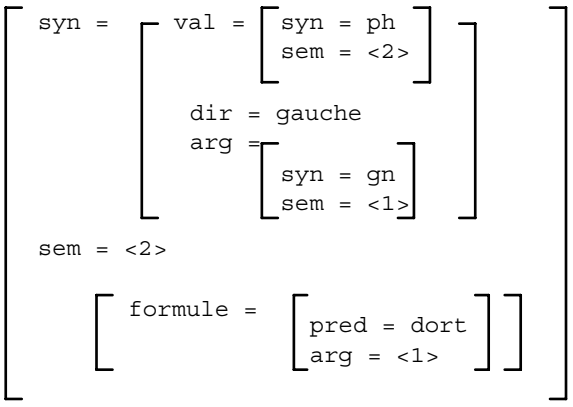


Fig 4.4: Construction d'une formule prédicative correspondant à "dort", instanciation du trait *foncteur* de la figure 4.3

Les DRS sont construites parallèlement à l'analyse syntaxique. Pour avoir une construction compositionnelle, correspondante à l'analyse gauche-droite de la phrase, quelques règles d'application supplémentaires ont été définies. Une DRS est représentée sous forme de liste. Référents du discours et conditions y sont mélangés mais facilement distinguables grâce à un typage approprié. La description d'une DRS sous forme de graphe suit le codage proposé par Klein [KLEI 86]. Des listes d'entrée (*in*) et de sortie (*out*) sont, en particulier, utilisées pour coder l'augmentation des structures au fur et à mesure de l'analyse. Ces DRS sont construites à partir des DRS partielles décrites dans le lexique. Les déterminants contiennent les informations essentielles qui donneront la structure générale de la DRS d'une phrase. La figure 4.5 montre une description d'une DRS partielle correspondante à l'entrée lexicale du verbe "voir" : la condition du type *voir(X1,X2)* est ajoutée en tête de la DRS courante. La présentation (approximative) sous forme DCG, utilisant les notations de liste "à la Prolog", rend peut-être encore plus explicite cet ajout simple.

```
verbe(V) --> [voir]
{ V:sem:in = [DRScour | DRSgen],
  V:sem:out =
[[ voir(V:syn:arg1,V:syn:arg2) |DRScour]|DRSgen] }.
```

```
[ sem = [ in = [ first = <1>
                 rest = <2> ]
          out = [ first = [ first = [ pred = voir
                                      arg1 = <3>
                                      arg2 = <4> ]
                            rest = <1> ]
                  rest = <2> ] ] ]
```

Fig 4.5: Construction d'une DRS partielle pour un verbe dans une notation 'à la DCG' et en STUF.(en représentation matricielle)

Nous ne présenterons pas d'exemple d'entrée lexicale de déterminant. La partie présentation du formalisme AVAG en donne des exemples assez proche de ceux de STUF.

Uszkoreit ne donne aucune justification aux choix de la DRT et ne détaille pas son traitement de la résolution des anaphores dans les articles cités. Il y est seulement fait mention de la simple correspondance entre la résolution de l'anaphore pronominale et la résolution de l'appartenance à une liste pour un référent du discours.

4.2.3. Eléments de comparaison

Les deux exemples de grammaires précédentes sont caractéristiques à plusieurs niveaux: façons de construire les DRS, par lambda réduction ou unification/augmentation de listes dans des structures de traits ; construction parallèle ou séquentielle des DRS ; type de grammaire (catégorielles et LFG) regroupant l'information dans le lexique.

Notre approche de la construction des DRS est voisine de celle d'Uszkoreit. Mais dans AVAG, il n'est pas utile de spécifier les structures de traits avant de les utiliser. Le compilateur se charge de regrouper les descriptions au fur et à mesure du traitement des entrées lexicales et des règles. Du fait de l'opération d'augmentation sur les structures de traits, nous pouvons nous dispenser de ces constructions de DRS sous forme de listes d'entrée et sortie. Référents du discours et conditions sont dissociés dans deux listes pour des raisons de clarté et d'efficacité dans la résolution d'anaphores.

Ces grammaires LFG et catégorielles sont sans doute plus condensées que la nôtre, du fait de cette concentration d'informations dans le lexique. Nous utilisons, nous aussi, des gabarits pour factoriser les descriptions lexicales, mais ne pouvons éviter la prolifération du nombre de règles de grammaires. Il semble, par contre, que cette approche guidée par le lexique fige un peu les stratégies d'analyse (Uszkoreit parle de stratégie montante, comme étant la seule possible), ce qui n'est pas le cas pour AVAG.

Il nous paraissait intéressant, par ailleurs, de présenter brièvement BUILDRS et les motivations de ses auteurs car nous reparlerons de leur système de résolution d'anaphores. Nous ne partageons évidemment pas leur point de vue sur cette dichotomie du développement de la DRT. Cette façon artificielle de séparer structuration du discours et interprétation s'explique peut-être par la focalisation sur des textes n'ayant aucun rapport avec un domaine particulier, et sur le choix d'exemples dont la couverture linguistique ne dépassent pas celle décrite par la mini-grammaire de Kamp.

4.3. ENVIRONNEMENTS DE DEVELOPPEMENT DE GRAMMAIRES

L'idée de développer des formalismes déclaratifs pour écrire des grammaires avec un souci d'implantation efficace, conduit tout naturellement à porter son attention sur les outils de développement de grammaires. Dans cette partie nous faisons un bref survol de ces outils en focalisant surtout notre attention sur ceux liés aux grammaires d'unification. Un environnement de développement de grammaire est l'un des constituants essentiels du poste de travail linguistique pour le traitement automatique de textes.

Les premiers environnements construits autour de grammaires d'unification n'ont d'abord été que des prototypes lents et peu efficaces simulant chacun une théorie particulière, tel ProGram [EVAN 85] outil de développement de GPSG. Avec PATR-II, qualifié quelquefois de "machine virtuelle linguistique" il est devenu possible d'encoder des grammaires de différentes natures. D-PATR [KART 86] est un bon exemple d'environnement convivial construit autour de PATR. Mais il reste, comme beaucoup d'autres, destiné aux étudiants/chercheurs pour s'exercer à la transcription de petites grammaires.

Des formalismes syntaxiques non basés sur les grammaires d'unification ont permis de développer des grammaires de large couverture: une des grammaires les plus connues étant DIAGRAM [ROBI 82] utilisée dans TEAM, l'interface en langage naturel pour bases de données. Mais ces approches étaient souvent procédurales, beaucoup de code LISP/Prolog se mêlant au formalisme lui-même. Des environnements complets s'appuyant sur des formalismes clairs ont été réalisés dans le milieu de la traduction automatique [VAUQ 85]. Mais, il semble qu'ils aient du mal à prendre en compte les nouveaux formalismes grammaticaux et les nouvelles stratégies d'analyse.

GDE (Grammar Development Environment) ([BOGU 88] et [BRIS 87]) est peut-être l'une des premières expériences de construction d'un environnement de développement de grammaires à large couverture basé sur l'approche "grammaire d'unification". Dans ce projet national britannique, le pari (tenu) était de développer en deux hommes/an une grammaire générale de l'anglais pour la mettre (avec la boite à outils utilisée pendant sa mise au point) à la disposition d'une large communauté du milieu TLN.

Les critères pris en compte pour améliorer la productivité du linguiste en charge de l'écriture de la grammaire étaient l'efficacité de l'implantation, la facilité d'utilisation, la robustesse de mise en oeuvre. L' analyseur morphologique et des stratégies d'analyse efficaces ont été mis au point séparément d'avec la grammaire. GED est composé d'un formalisme de spécification de grammaire de type GPSG et d'un environnement de développement. Le formalisme est présenté comme la méta-grammaire dans laquelle sont décrits les "universaux" de la langue. La grammaire objet est celle qui est effectivement exécutée par l'analyseur après compilation de la méta-grammaire.

Trois points sont mis en évidence dans l'environnement :

- développement rapide et incrémental : possibilité de ne recompiler que les règles modifiées après une mise-à-jour de la grammaire, bonne correspondance entre méta-grammaire et grammaire objet pour localiser facilement les erreurs[1] ,...
- mise au point interactive: possibilité d'analyse sélective et de génération (ou plus précisément de synthèse, car on n'utilise exactement la même grammaire dans deux sens différents) de parties de la grammaire. Par exemple, une description partielle de règle suffit à permettre la synthèse de phrases d'une longueur donnée. possibilité d'affichage selectif de règles activées lors de l'analyse, suivi en temps réel des arbres construits,...
- portabilité: pas d'environnement graphique dans la version de base, programmation LISP standard.

Les spécifications du poste de travail de ACTES recouvrent partiellement celles de GED (mais pour un projet de taille beaucoup plus modeste). La méta-grammaire est AVAG et la grammaire objet est de type DCG. Développement de stratégies d'analyse/synthèse et développement du formalisme ont été clairement séparés. Notre formalisme a été testé sur une grammaire plus large du français [CHAN 87] que celle présentée ici dans le cadre du projet ACTES. L'environnement de développement sera présenté plus loin.

---

[1] La méta-grammaire offre généralement des possibilités syntaxiques permettant de factoriser les règles (cf les méta-règles en GPSG). Lors de la compilation, il y a expansion du nombre de règles. Le rapprochement entre règles objet et règles source est alors problématique.

L'aspect convivialité du système ne semble pas avoir été pris en compte dans GED. Si un des objectifs du poste de travail est de dégager le linguiste de préoccupations d'implantation, alors cet aspect n'est pas à négliger. En particulier une interface graphique contribue beaucoup à l'amélioration de la qualité du travail de l'utilisateur. Mais elle accroît sensiblement les problèmes d'implantation de l'environnement. L'argument sur la portabilité est un peu spécieux,.car des normes standards graphiques (X-Windows, par exemple) commencent à apparaître.

La problématique du poste de développement de grammaire en est encore à ses débuts[2]. Si l'on fait l'hypothèse que certains formalismes grammaticaux sont arrivés à maturité et que l'on peut construire des environnements de développement autour, il faudra se donner les moyens de mesurer la qualité d'un environnement sur des critères linguistiques et non linguistiques. De tels critères existent aujourd'hui dans la branche des interfaces en langage naturel [BATE 87].

4.4. SYSTÈMES DE TRAITEMENT AUTOMATIQUE DE TEXTES

Traiter automatiquement du texte et disposer d'un environnement correspondant implique la prise en compte de bien d'autres facteurs que le développement d'une grammaire. En particulier, il faut représenter les connaissances liées au domaine du texte, la structuration du texte, les phénomènes qui assure sa cohésion (dont l'anaphore n'est qu'un cas particulier) et sa cohérence[3]. Dans cette partie, nous présentons succinctement quelques systèmes de traitement automatique de textes qui font référence, puis portons notre attention sur ceux qui utilisent grammaire d'unification et/ou DRT[4].

---

[2] Les premières réalisation d'outils linguistiques, et particulièrement d'outils de développement de grammaires, datent des origines de la linguistique informatique: cf le langage COMIT créé en 1958 [GAZD 87b] dont les spécifications ressemblent curieusement à celle de PATR-II qui prétendait pourtant à l'originalité. Mais ces outils ont montré rapidement leurs limites et ont surtout été utilisés par des communautés de chercheurs très restreintes numériquement. Les choses sont peut-être en train de changer avec les outils développés autour des grammaires d'unification et supportés par des projets nationaux, comme GED.

[3] La liste n'est pas exhaustive. Pour trouver une bonne introduction sur les phénomènes linguistiques qui caractérisent un texte et le différencie d'une simple suite de mots ou de phrases sans relation entre elles, on peut se reporter au livre de de Beaugrande , *Introduction to Text Linguistics* [BEAU 81].

[4] Signalons un système de compréhension de textes utilisant une grammaire d'unification réalisé en France par Rousselot [ROUS 84].

4.4.1. Systèmes généraux de compréhension de textes

Le traitement automatique de textes de volume important et portant sur des domaines étendus (problème crucial en aéronautique, compte tenu de l'abondance de la documentation technique) ne concerne principalement que la manipulation de documents ([CHRI 87] pour la documentation aéronautique), le stockage, la recherche, l'indexage par mots-clés. Les systèmes prenant en compte certaines données linguistiques (morphologie, syntaxe), couplés en général avec des données statistiques, utilisent assez peu les techniques d'Intelligence Artificielle quand ils veulent privilégier la robustesse par rapport à la compréhension en profondeur ([RIAO 88] pour un aperçu global).

Les recherches en compréhension de textes sont récentes. Elles ne portent, bien sûr, que sur des domaines techniques limités, à sémantique restreinte, domaines dans lesquels le langage utilisé est lui aussi restreint (télex bancaires, messages de marine, maintenance de matériels, annonce de journaux sur les reprises de sociétés). La taille des textes analysés est aussi limitée. Mais sous réserve de ces limitations, des prototypes robustes permettant une analyse fine de la sémantique commencent à voir le jour.

Deux grands courants, s'inspirant de théories linguistiques et de modèles de représentation des connaissances éprouvés en Intelligence Artificielle se partagent ces réalisations. Leur but est de traduire des textes ou des messages sous des formats internes permettant ensuite l'interrogation et la recherche d'informations.

Un des courants [KITT 86] s'attache à étudier le sous-langage du domaine spécialisé. Des analyseurs robustes peuvent être construits tant que les textes utilisent le vocabulaire et les structures linguistiques du sous-langage. Le système PUNDIT [HIRS 88] permet même l'acquisition interactive de connaissances sémantiques en couplant formats internes syntaxiques du sous-langage avec un module de représentation hiérarchique des connaissances. Un système de résolution de référents et de recherche du focus assure la liaison entre les phrases. Le système utilise une grammaire dérivée de la grammaire générale de [SAGE 81].

L'autre approche, plutôt que de s'appuyer sur les structures linguistiques du domaine, utilise les contraintes sur les connaissances du domaine. La liaison entre les différentes parties du texte est prise en charge par une structure de représentation tirée des scripts et des Dépendances Conceptuelles de Schank. Ce système permet à un instant donné de l'analyse, de faire des prédictions sur la suite du discours [LYTI 86]. Les données linguistiques du texte ne sont utilisées qu'à titre de compléments d'informations [RAU 88].

Les systèmes développés dans les deux courants sont composés de modules différents (grammaires issues de théories linguistiques, générateurs de phrases, langage de représentations des connaissances, organisation de lexiques spécialisés, résolution d'anaphores), développés avec l'objectif d'être portables à des domaines techniques variés. La prise en compte de la structuration des textes sous formes de chapitres, titres, paragraphes, ainsi que la représentation des quantificateurs n'apparait explicitement dans aucun des travaux mentionnés. De même, les allusions faites au problème de la vérification de cohérence lié à un système de résolution (problème très important dans le domaine particulier des textes de spécifications) sont peu fréquentes. [BIEB 87] aborde ce point dans le cadre du développement d'un système d'aide à l'élaboration d'un cahier des charges.

4.4.2. Systèmes utilisant la DRT

Les systèmes précédents s'attachent à traiter des textes réels. Leurs auteurs discutent de la qualité des résultats obtenus par rapport aux objectifs initiaux d'efficacité. Dans les systèmes utilisant la DRT, il semble que les préoccupations des chercheurs soit d'une autre nature. Il est difficile de se faire une idée des types, quantité, difficultés des textes traiter effectivement. Certains problèmes plus ambitieux tels le temps, les pluriels, la prise en compte de l'espace sont abordés, mais n'ont souvent pas été intégrés (même partiellement) dans ces systèmes. Dans la présentation des deux systèmes nous nous attacherons à identifier la nature des utilisations de la DRT.

4.4.2.1. Le système LEX

Le projet "Linguistics and Logic Based Legal Expert System" ou LEX ([GUEN 86], [LEHM 88]) a regroupé pendant trois ans un nombre important de chercheurs de l'université de Tübingen et du centre scientifique IBM de Heidelberg en Allemagne sur des problèmes d'analyse du discours, représentation des connaissances, techniques de démonstration, formalisation des lois.

Leur travail peut grossièrement se décomposer en deux étapes bien distinctes. D'abord une large étude linguistique des textes de loi germaniques a été menée avec le souci d'identifier les problèmes linguistiques, les connaissances expertes ou implicites (de sens commun). Une base de données lexicales a été créée et couvre une grande partie du vocabulaire des textes. Des spécifications ont été faites sur ce que devrait être un système d'extraction de connaissances à partir des textes de lois permettant de dialoguer avec un expert du domaine.

Dans un deuxième temps, le champ d'application a été fortement restreint et un prototype a été développé afin de permettre à un expert de la législation routière de décrire un cas d'accident sous forme de texte en langage naturel puis de dialoguer avec le système sur l'interprétation du cas en rapport avec les lois. Les difficultés abordées dans ces textes concernent principalement la référence contextuelle et certains problèmes liés au temps et aux expressions temporelles. Le système utilise une grammaire précédemment développée pour une interface avec une base de données, des DRS, la base de données lexicales et un démonstrateur de théorèmes.

Les DRS ne servent pas seulement de représentations sémantiques intermédiaires sur lesquels opèrent le démonstrateur et le module de résolution d'anaphores, mais aussi de formalisme de représentation des connaissances. Ainsi les règles exprimant les connaissances de sens commun et celles codant les connaissances expertes sont décrites par des DRS. Le démonstrateur n'utilise pas de méthodes de résolution habituelles, mais la méthode des tableaux. La déduction opère directement sur les DRS et permet en cas d'échec d'exhiber des contre-exemples. L'ensemble de l'approche est donc relativement uniforme. Le module de résolution d'anaphores est présenté au chapitre 7. On notera que les conditions d'accessibilité logiques décrites par la DRT ne sont qu'une des sources de contraintes prises en compte au côtés de contraintes morphologiques, syntaxiques, pragmatiques.

Discussion

Ce projet recoupe par bien des côtés la problématique de ACTES. L'objectif à long terme est l'extraction de règles pour un système expert sur la législation allemande. Mais les auteurs, après avoir constaté que l'expression des règles se faisait souvent dans les textes réels par raffinements successifs, ont d'emblée écarté ce problème. Cet a priori est très fort car la prise en considération de ces processus non-monotones nous paraît centrale dans des textes d'expertise (cf chapitre 8). Les objectifs du projet nous semblent un peu confus. En effet, les auteurs se placent dans le cadre du paradigme 'système expert' en prétendant apporter une contribution originale par leur approche logique ([LEHM 88] pp 119). Ils confondent compréhension de textes d'expertise et systèmes experts (nous distinguons dans ACTES ces problèmes). Les chercheurs du second domaine sont surtout préoccupés par la gestion de grandes quantités de règles et donc recherchent des structures appropriées et des méthodes de déduction efficaces. Dans LEX le nombre de règles manipulées est très limité, de l'aveu même des auteurs, et rien ne prouve que les DRS et la méthode des tableaux puissent être étendues sans problème. Le passage d'un système expert de taille limité à celui permettant de gérer une application grandeur réelle est en effet l'un des principaux points d'achoppement.

Malgré le désir des auteurs de considérer la DRT comme un langage de représentation des connaissances "universel"[5], il reste à montrer quelles sont exactement les réponses originales qu'elle apporte et ce qu'elle fait de mieux que les autres langages. Ainsi les connaissances descriptives sur les concepts du domaine (de type généralisation/spécialisation,...) ne sont pas codées dans des DRS et sont beaucoup plus limitées que celles traitées justement en IA. De même les définitions de l'accessibilité logique de la DRT sont implicitement reconnues comme limitées mais non discutées explicitement.

Le débat sur tous ces points est un peu faussé. Les auteurs justifient essentiellement l'utilisation de la DRT par sa proximité avec la logique des prédicats et s'intéressent donc à sa puissance d'expression, à son interprétabilité, aux notions de vérité, de décidabilité ([GUEN 86] pp 40),... mais ignorent tous les travaux réalisés sur ces points dans les langages d'IA (cf la présentation de BACK [LUCK 86], le langage que nous utilisons, et [PATE 89] qui renvoie à des travaux plus anciens).

4.4.2.2. Le projet ACORD

Le projet ESPRIT ACORD s'intéresse à la construction et à l'interrogation de bases de connaissances à partir de textes en langage naturel et de graphiques [ACOR 89]. Arrivé prochainement à échéance, il aura regroupé pendant cinq ans des chercheurs des universités d'Edimbourg, Stuttgart, Clermont-Ferrand, des centres de recherche industriels de la CGE, de Triumph-Adler et de Bull.

Le domaine d'application choisi pour la réalisation du prototype est celui des transports routiers. Aucun texte réel n'a été considéré, il n'y pas donc pas eu d'étude linguistique du domaine particulier des transports. Les auteurs se sont intéressés, au contraire, à décrire des grammaires générales des langues anglaise, allemande et française. Certains phénomènes linguistiques comme celui des locutions de lieu, de temps, certains types de pluriels ont été pris en compte.

L'utilisateur peut mettre à jour et questionner la base en dialoguant avec le système comme dans l'exemple suivant:

---

[5] au sens où la DRT permettrait d'aborder tous les problèmes sémantiques auxquels se heurtent les chercheurs en Intelligence Artificelle [GUEN 86].

```
- Camion1 et camion2 sont à Paris.
- Camion3 et camion4 sont à Stuttgart.
- Ils sont chargés de câbles.
- Camion1 et camion2 transportent des terminaux.
- Que transportent camion1 et camion2 ?
- Des câbles et des terminaux.
```

L'utilisateur peut également (c'est une des originalités du système) mettre-à-jour la base par des actions de désignation/déplacement d'objets graphiques sur l'écran. Ainsi le déplacement d'un camion, d'une ville-dépôt à une autre, génère une requête (équivalente à la formulation de cette action en langage naturel) et met à jour les stocks des dépôts de chacune des villes.

Le système comprend trois grammaires qui sont utilisées en analyse ou génération, une base de connaissances, un démonstrateur de théorèmes, un module d'interface graphique, un module de gestion du dialogue. Les grammaires sont de type LFG ou UCG (cf la présentation de UCG au chapitre 2). Le formalisme de représentation sémantique, INL, est unique pour tout le système: graphique, langage naturel, démonstrateur. Il s'inspire fortement de la DRT. La figure 4.6 présente l'architecture du système.

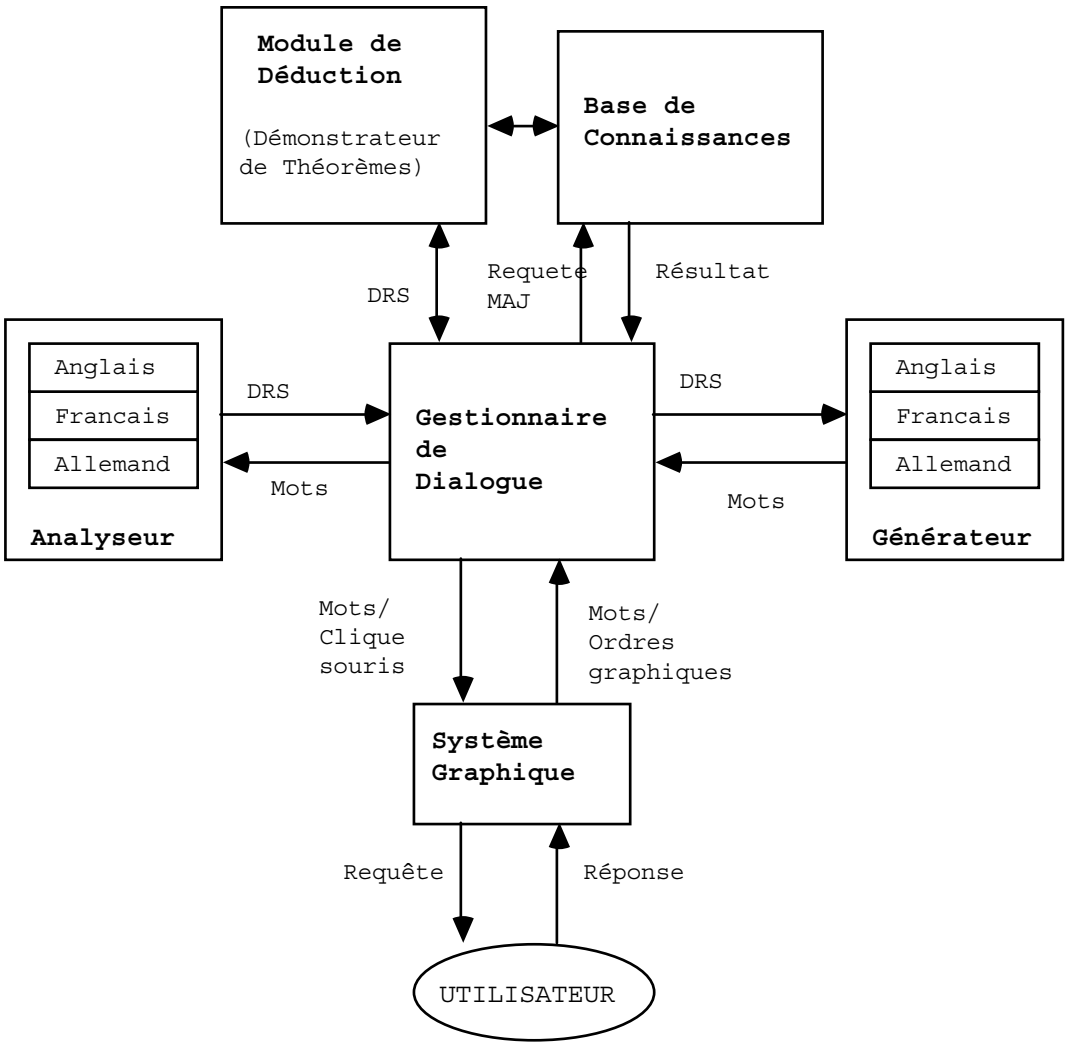


Fig 4.6: Architecture d'ACORD.

Détaillons maintenant la partie qui concerne la représentation des connaissances et leur gestion (cf figure 4.7) [HEYE 88]. Les connaissances terminologiques sont décrites dans une structure hiérarchique de frames. Là sont représentés, par exemple, les connaissances sur un camion, le fait qu'il est conduit par un chauffeur, a un moteur, ... Le module Raisonneur Conceptuel gère les mécanismes d'héritages ainsi que des connaissances générales du genre "tous les objets concernés par un déchargement sont situés dans le même lieu". Ces connaissances s'expriment sous forme de règles Prolog. Toutes ces connaissances sont considérées comme stables et ne peuvent être mises à jour.

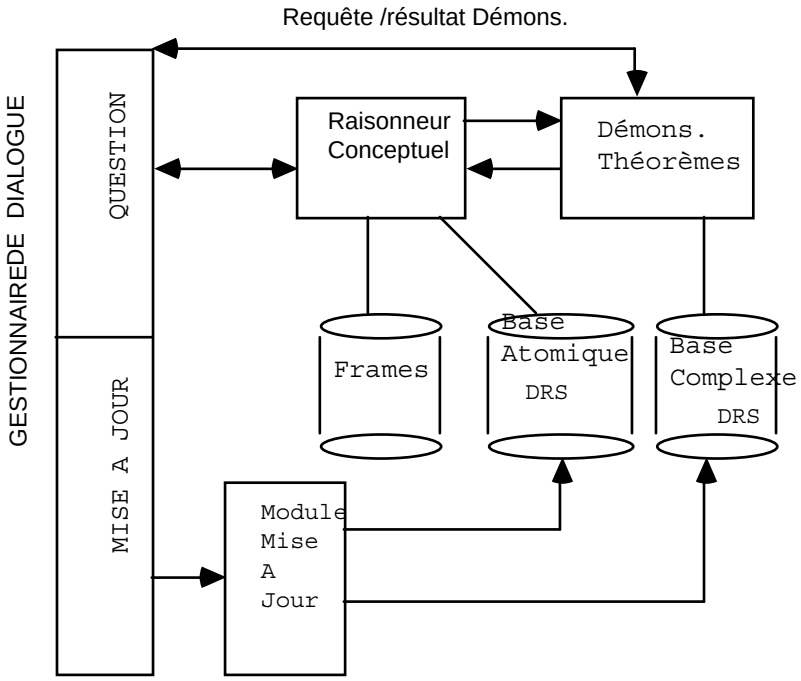


Fig 4.7: Répartition et gestion des connaissances dans ACORD

Les requêtes de l'utilisateur, traduites en INL, sont décomposées en faits et règles INL. Elles correspondent à des affirmations ou questions du genre "camion1 transporte du vin", "tous les camions transportent du vin", "qui conduit camion1 ?". Ces connaissances peuvent être vraies ou fausses. C'est le démonstrateur de théorèmes qui tranchera. Il est, lui aussi, basé sur la méthode des tableaux [OMMA 88] et opère des déductions sur les règles INL, en accédant, si besoin est, au Raisonneur Conceptuel. Les faits INL ne sont accessibles qu'à partir du Raisonneur afin d'éviter des appels réciproques infinis entre Raisonneur et Démonstrateur.

Discussion

Un travail syntaxico-sémantique (sémantique au sens restreint, sans référence aux connaissances du domaine) important a été réalisé lors de l'élaboration des différentes grammaires. Les représentations sémantiques intermédiaires sont très homogènes entre toutes les composantes du système. Un partage intéressant des connaissances a été fait. Il rappelle par bien des aspects la dichotomie connaissances intensionnelles-connaissances assertionnelles des langages hybrides, comme BACK, dans lesquels les premières sont représentées et gérées dans la boite terminologique (Tbox), et les secondes dans la boîte assertionnelle (Abox).

Mais, alors que le projet aurait pu donner lieu à un rapprochement intéressant entre sémantique formelle/DRT et Intelligence Artificielle, chaque partie a été développée de façon indépendante et les exemples de dialogue impliquant tous les modules du système n'apportent rien de très nouveau. La couverture syntaxique est bien plus large que ce que le système est en mesure de "comprendre" effectivement. Les types de requêtes que le système peut traiter[6] sont assez comparables à celles généralement rencontrées dans les interfaces LN-base de données. Tout se passe comme si d'un côté, on pouvait introduire des notations subtiles pour représenter l'intentionnalité, le temps, les pluriels dans les INL [BETH 89] alors qu'une autre partie du système ne pourrait pas intégrer ces phénomènes et traiterait les INL-DRS comme de simples formules logiques traditionnelles.

Le type de texte abordé ne permet pas de savoir si la DRT représente bien la sémantique d'un discours. Les problèmes de quantification occupent une place de choix dans ces requêtes, alors que ces problèmes deviennent secondaires dans beaucoup de textes (comme ceux de ACTES, mais aussi ceux traités par ATRANS, SICSOR, PUNDIT). Les problèmes liés au domaine, au suivi du discours, à la pragmatique sont très secondaires dans ACORD. Ils n'interviennent pratiquement pas dans la résolution des anaphores. Si ces problèmes devaient être pris en compte, le programme de résolution d'anaphores pourrait-il resté cantonné dans la partie Analyseur du système ? Que deviendrait la répartition des connaissances linguistiques/non-linguistiques (cf discussion sur les types au chapitre 2) et l'architecture même du système ?

4.5. UN ENVIRONNEMENT POUR LA COMPRÉHENSION DE TEXTES

Nous voudrions terminer la présentation de ces systèmes en disant quelques mots d'un des projets les plus ambitieux en TLN qui rassemble tous les paradigmes rencontrés jusqu'à présent. Il s'agit du projet de recherche en compréhension de textes du second laboratoire de l'ICOT au Japon ([ICOT 86], [ICOT 87]). Commencé en 1982, avec le début des recherches sur les ordinateurs de la "cinquième génération", il s'est développé suivant deux axes principaux: élaboration d'une boîte à outils linguistiques *Language Tool Box* (LTB) à partir de laquelle est construit un système de compréhension de textes, Discourse Understanding Aimed at Logic Based Systems (DUALS).

En voici les principales caractéristiques :

- au niveau matériel: machine "Prolog" séquentiel PSI, et bientôt machine "Prolog" parallèle PIM.
- au niveau langages:

[6] nous ne parlons pas ici de la partie sémantique de l'interface graphique qui est elle originale.

-- langage machine dérivé de Prolog ESP pour la machine séquentiel et GHC pour la parallèle.

-- langage de la boite à outils LTB: CIL extensions de Prolog vers la programmation logique avec contraintes [JAFF 87].

- au niveau théories linguistiques:

-- grammaires d'unification, Théorie des Situations, Sémantique Situationnelle [BARW 83].

L'architecture du système de compréhension de textes est détaillée en figure 4.8. On remarquera que les chercheurs de l'ICOT veulent aborder tous les problèmes liés au discours et en particulier prendre en compte les actions et intentions des locuteurs qu'ils représenteraient dans le cadre de la Théorie des Situations et géreraient avec des techniques de planification.

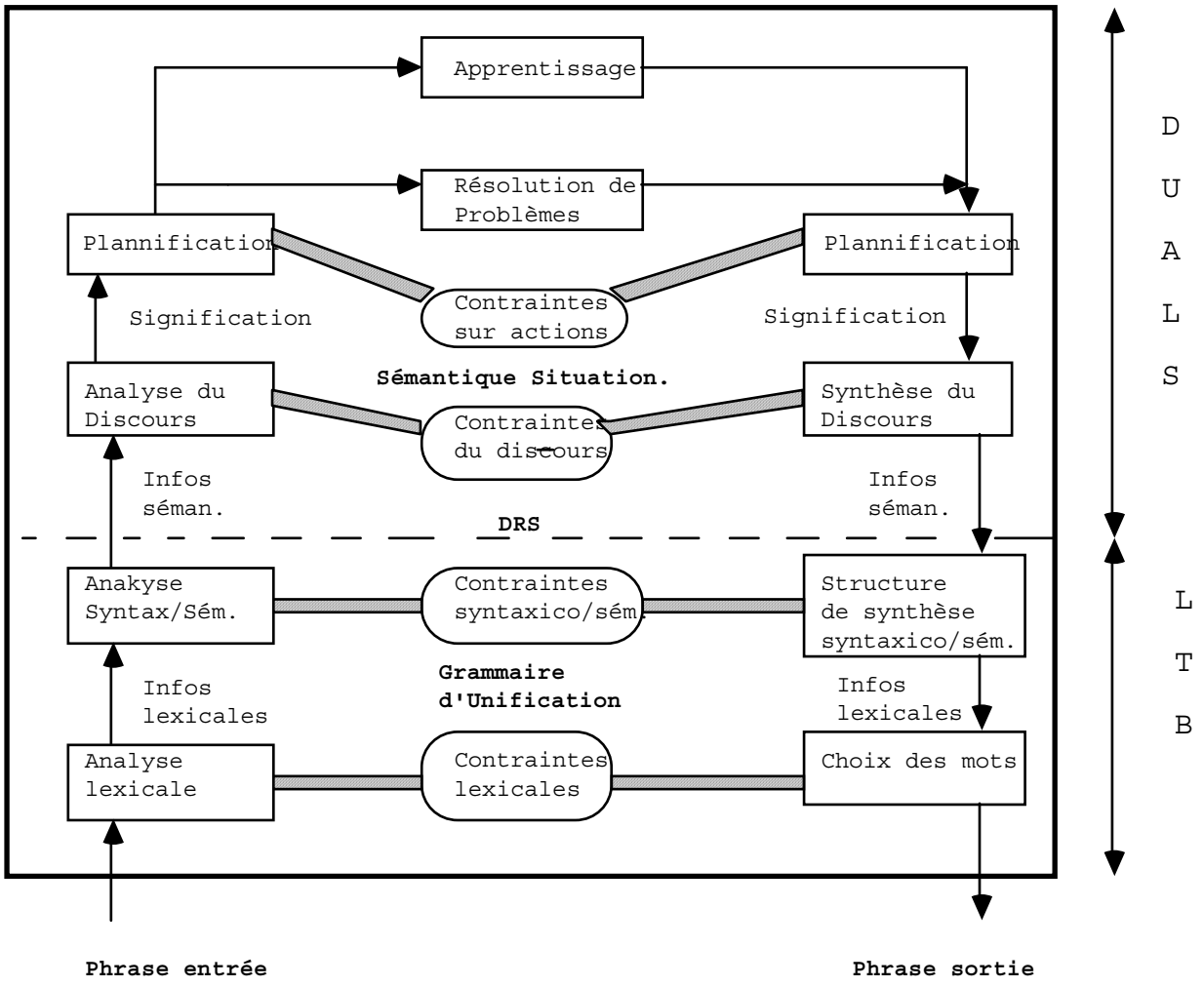


Fig 4.8: Architecture du système de compréhension de textes de l'ICOT

La configuration de l'environnement de développement LTB [SUGI 88] est détaillé en figure 4.9. Chaque partie dispose de sa propre boite à outils pour la mise au point, l'édition. Les analyseurs, lexical et grammatical, peuvent fonctionner en série ou parallèle. L'analyseur grammatical a une stratégie montante couplée à une stratégie descendante pour la prédiction.

Toutes les parties sont écrites en CIL. Il s'agit d'une extension de Prolog, comprenant des termes partiellement décrits (PST) et des mécanismes de co-routinage comme le *freeze* de Prolog-II. Une PST se présente sous la forme d'un ensemble de couples attribut-valeur. Les structures de traits des grammaires d'unification peuvent donc se décrire directement en utilisant les structures de données primitives du langage. Un ajout de mécanisme d'héritage reposant sur des types est même prévu ! Les contraintes linguistiques s'expriment sur les termes PST à l'aide des mécanismes de contraintes de CIL. Le *freeze* n'en est qu'un exemple, puisque prochainement des contraintes booléennes seront intégrés dans CIL comme dans les langages CLP et CHIP [DINC 87].

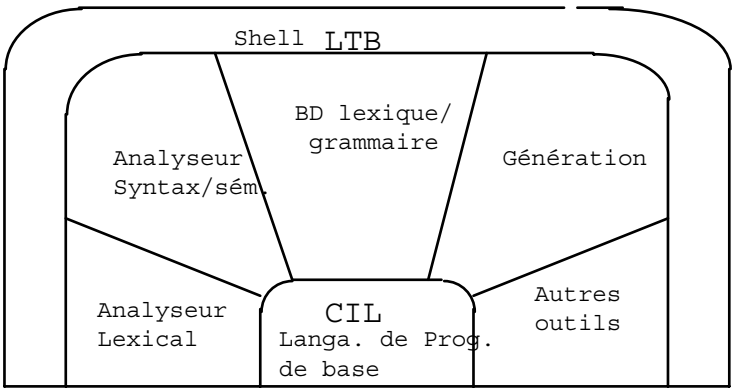


Fig. 4.9: Configuration de l'environnement de développement LTB.

Tous les mots-clés sont donc présents dans ce projet, ceux caractérisant les derniers travaux en programmation logique, en grammaire d'unification, en traitement du discours. Qu'en est-il des réalisations ? Une telle méthodologie de recherche est-elle efficace ? Difficile de répondre à ces questions au regard des documents dont nous disposons.

La partie la plus avancée concerne l'environnement LTB. Le fait de disposer de formalismes grammaticaux reposant directement sur des structures de langage de programmation répond de façon concrète aux problèmes que nous avions évoquées dans la discussion du chapitre 2.

L'ambition des auteurs du projet est de traiter des textes japonais d'un niveau de sixième année d'école. Qu'entendent-ils par là ? Les textes généraux utilisés dans les classes sont sans doute d'un niveau de complexité supérieur à ceux d'un domaine technique pointu, mais des précisions font défaut sur ce point. Très peu de choses sont dites sur la Théorie des Situations, sinon qu'elle n'est pas encore à maturité (il n'existe pratiquement pas d'implantation à une échelle significative de cette théorie aujourd'hui dans le monde). Alors pourquoi ce choix ?

Affaire à suivre ... pour la réponse à toutes ces questions !

5. PRÉSENTATION DE ACTES

Ce chapitre introduit la deuxième partie de cette thèse dans laquelle nous allons développer nos travaux accomplis dans le cadre du projet ACTES.

Nous commençons par donner une idée générale du contenu et de l'organisation des textes de spécifications des logiques d'alarme. Puis nous indiquons, suivant quels critères, après une étude linguistique générale de tous les textes du domaine, nous avons sélectionné un sous-ensemble pour tester notre prototype de poste de travail linguistique. Les caractéristiques principales du corpus de ces textes sont dégagées, mettant ainsi en lumière les principaux problèmes auxquels nous étions confrontés. Un texte caractéristique est donné en illustration. Il servira de support à nos exemples dans les chapitres suivants.

Puis l'architecture générale du système ACTES est présentée. Chacun de ces modules sera détaillé par la suite. Enfin, un scénario d'utilisation éclaire le cheminement procédural que doit suivre l'utilisateur de notre système et explicite ainsi les liens logiques entre les différents modules[1]

5.1. CARACTÉRISTIQUES DU CORPUS

L'ensemble des textes de spécifications des logiques d'alarme de processeurs embarqués sur un avion se divise en plusieurs sous-pavés, concernant chacun un système avion (Trains et Trappes, Freinage, Moteurs,...). Les textes contenus dans un sous-pavé sont structurés en chapitres. Chaque chapitre spécifie un traitement (assez proche de la notion de programme informatique). L'ordre des chapitres n'est pas indifférent, il correspond à l'ordre dans lequel les processeurs temps-réel des systèmes avion traitent séquentiellement les informations.

5.1.1. Limites de l'étude

Une étude linguistique détaillée a été menée avec Corinne Fournier sur plusieurs sous-pavés. On a fait une double distinction : entre les chapitres (ou sous-chapitres) en langage naturel où la notion de temps intervient et ceux où elle n'intervient pas.

[1] Corinne Fournier, ingénieur de recherche chez AMD-BA, et moi-même, étions les principaux artisans du projet. Corinne a réalisé la plus grande partie de l'analyse linguistique du corpus des textes et a écrit les grammaire et lexique du prototype en utilisant AVAG. J'ai, pour ma part, conçu l'architecture générale du système, spécifié et développé le formalisme AVAG, ainsi que chacun des modules du système, développé l'interface graphique de ACTES et étudié le problème de la traduction des textes en logique des défauts. Catherine Rodriguez m'a aidé dans les tâches d'implantation. La description des connaissances du domaine a été menée conjointement par Corinne et moi.

Les tableaux traduisent l'attribution de valeurs, dans certaines conditions aux informations de sortie (par recopie à partir d'une entrée ou par forçage) ou définissent les valeurs limites de paramètres[2]. Les tableaux sont une forme de notation concise et claire pour exprimer des conjonctions de conditions et d'actions. Ces données pourraient être traduites en langage naturel.

De nombreux chapitres font intervenir la **notion de temps**. Ces textes décrivent des traitements complexes. Un traitement est une succession de cycles. La détermination de la valeur d'une info d'entrée est souvent l'objet d'un traitement de plusieurs cycles ; une info d'entrée connaît plusieurs transitions de valeurs. Deux types de problèmes se posent, qu'il ne nous a pas semblé envisageable de traiter (ou même de tenter de traiter) dans le cadre de ce projet : du point de vue informatique, la modélisation d'une application temps-réel, et du point de vue linguistique, l'analyse et le représentation formelle du temps.

On a distingué un ensemble de **textes homogènes dans lequel le temps n'intervient pas**, et dont la sémantique est un sous-ensemble de celle des textes où le temps joue un rôle. La relative simplicité de ces textes permet de décrire formellement ce qu'est un traitement sous forme de règles conditions/actions, en manipulant les valeurs des infos de validité. Aucune notion de cycle de traitement ne s'y rattache. En fait, au cours d'un traitement, une information, avec sa valeur et sa validité, est considérée dans un état unique, elle ne connaît pas de transition. Ces textes, plus simples, ne sont pas pour autant dénués d'un intérêt pour l'étude linguistique. De nombreux phénomènes apparaissent (noms composés, référence, métonymie, ...) que l'on a étudiés sur un corpus plus large que celui retenu pour le prototype.

Se limiter à l'analyse de ces textes n'est pas adopter une démarche étroite. Une étude linguistique assez vaste, le développement d'outils généraux devraient permettre de traiter par la suite d'autres textes, concernant le même domaine.

5.1.2. Un texte prototypique

Voici le premier chapitre du sous-pavé Trains et Trappes, texte dont sont tirés certains des exemples que nous donnons par la suite.

---

[2] La signification de ces termes techniques sera détaillée dans les chapitres suivants et particulièrement dans le chapitre 8.

1. Les infos de sortie:

Alarme-gear-not-down et Disc-gear-not-down sont positionnées à alarme si simultanément :

- train-avant-verrouillé-bas est non-verrouillé-bas
- train-gauche-verrouillé-bas est non-verrouillé-bas
- train-droit-verrouillé-bas est non-verrouillé-bas
- vitesse-conventionnelle est inférieure à Vmini
- angle-manette-gauche est inférieur à angle-manette-mini
- angle-manette-droite est inférieur à angle-manette-mini
- hauteur-radio-sonde est inférieure à HRS-mini.

Dans les autres cas, les sorties sont positionnées à non-alarme.

Si la vitesse-conventionnelle est invalide, on utilise la vitesse-sol. Si la vitesse-sol est invalide également, on considère que la condition sur la vitesse est remplie, c'est-à-dire inférieure à Vmini.

Si la hauteur-radio-sonde est invalide, on utilise l'altitude-baro-inertielle. Si l'altitude-baro-inertielle est invalide également, on considère que la condition sur la hauteur est remplie, c'est-à-dire inférieure à HRS-mini.

Si un paramètre est invalide, on considère que la condition sur le paramètre est remplie.

Nota: Les deux infos de sortie sont forcées à non-alarme si l'info d'entrée, Val-infos-TT-redondées, est à invalide.

Remarque: On positionne l'info interne Train-non-sorti simultanément et à la même valeur que la sortie Gear-not-down.

Fig 5.1: Exemple de texte concernant le train d'atterrissage

La première partie de ce texte signifie à peu près ceci:

- les trains ne sont pas sortis
- la vitesse est petite
- l'angle manette est petit
- l'altitude est faible

=> alarme: l'avion est près du sol et ne peut pas atterrir sans ses trains sortis

L'angle manette représente la commande sur la poussée moteur. Il y a une manette par moteur. ; un angle petit indique une poussée commandée faible.

La signification des paragraphes concernant les conditions de validité sera détaillée dans le chapitre concernant l'extraction des règles.

5.1.3. Caractéristiques principales du corpus

Les caractéristiques principales du corpus que nous avons à traiter sont les suivantes:

Syntaxe des textes

Les phrases sont des phrases déclaratives simples, sans relative, sans négation. Ces phrases expriment des règles, elles sont souvent de la forme "si ... alors ...".

Noms d'informations

On a relevé les noms des informations et les noms de leurs valeurs pour en faire l'analyse morphologique, syntaxique et sémantique. On a également étudié les rapports syntaxiques entre les informations et les valeurs. Les résultats de cette étude linguistique complète ne sont pas pris en compte pour la réalisation du prototype. Le traitement détaillé des noms d'information n'aurait de sens que si l'on désirait enrichir automatiquement la base de connaissances du système. C'est un travail complexe (cf [ISAB 84]) qui n'entre pas dans le cadre du projet. On considère donc un nom d'information comme un tout insécable.

Ambiguïtés sémantiques

On ne trouve pas d'ambiguïtés sémantiques. Les mots techniques sont employés dans un sens unique.

Connaissances pragmatiques (ou extra-linguistiques)

La modélisation des connaissances du domaine est indispensable pour la compréhension du texte. Même un humain, non expert du domaine, ne peut comprendre ces textes sans avoir recours à ces connaissances souvent implicites. Nous distinguons deux types de connaissances pragmatiques, celles sur les objets du domaine, notamment les informations, décrivant leur nature (type, valeurs, validité), leurs inter-relations, et les métaconnaissances liées à la structure des textes.

Structure des textes

La structure du texte apporte des métaconnaissances sur les informations et la composition des règles.

Le titre peut se présenter sous des formes et avec des contenus différents. Ce peut être: un vrai titre (il résume ce qui suit), le nom d'une sortie, d'une entrée, le début d'une phrase, une référence à un chapitre précédent. Il apporte des informations auxquelles il est souvent fait référence par la suite. On décide, pour l'instant, de toujours attribuer au titre le rôle d'introduire le nom de la sortie traitée par le chapitre en question.

L'étude des **notas et remarques** permet de dégager une régularité presque toujours observée dans le rôle que jouent ces sous-chapitres. Le nota indique souvent la valeur de forçage de la sortie en fonction de la validité des entrées. La remarque éventuelle spécifie le positionnement d'informations internes à la même valeur que la sortie concernée.

L'analyse des **chapitres** de plusieurs sous-pavés nous a permis de définir une structure stable et de corriger les textes qui nous sont présentés. Un chapitre sera divisé en cinq parties: titre, règle principale de positionnement, contrôles sur validité, forçage, effet de bord. Connaître la structure d'un texte est indispensable pour l'analyse de textes. Un système de traitement automatique du langage naturel doit savoir de quoi on parle à tel endroit du texte. Ces informations de structuration seront utilisées pour la génération des règles de défaut.

Référence et anaphores

La **référence** est un problème crucial pour l'analyse automatique de textes. De nombreux mots, comme les pronoms (personnels, relatifs, possessifs, démonstratifs), et même le simple article défini, font surgir des ambiguïtés qu'il faut résoudre. Parfois, les informations syntaxiques suffiront, parfois il faudra faire appel à la sémantique et à la pragmatique. Nous évoquons ici certains des problèmes que pose l'article défini dans nos textes.

L'article défini peut introduire un nouvel objet, comme dans le titre "les infos de sortie ...". Il peut également faire référence à un objet déjà introduit: "la vitesse conventionnelle". Dans "la condition sur la hauteur", il est fait référence à la condition émise dans la règle principale, condition portant sur une information particulière "hauteur-radio-sonde" dominant l'information "altitude-baro-inertielle". Ces deux informations relèvent du concept de "hauteur".

L'article pluriel pose un problème supplémentaire. Fait-on référence à un ensemble pris dans sa globalité "les sorties" (toutes les sorties) ou à une partie de cet ensemble? De quels éléments parle-t-on alors? Comment sont-ils caractérisés et reconnaissables?

La phrase "si un paramètre est invalide ..." Il s'agit en fait des informations dont on n'a pas encore parlé dans les conditions générales d'applicabilité de la règle. Il ne s'agit pas des paramètres auxquels il a été fait référence par les concepts de vitesse et de hauteur. Il s'agit des autres paramètres, c'est-à-dire l'angle-manette-gauche et l'angle-manette-droit. Cette règle est en fait une règle plus générale que les deux précédentes.

5.2. ARCHITECTURE DU SYSTÈME ACTES

Pour la première partie du travail d'analyse de textes, reconnaissance des constituants du texte et traduction sous une forme sémantique intermédiaire, nous avons développé **un poste de travail linguistique**. L'idée est d'offrir, au linguiste chargé d'écrire lexique et grammaire, un environnement suffisamment riche pour exprimer les connaissances linguistiques contenues dans les textes et les connaissances liées au domaine tout en le déchargeant le plus possible des préoccupations d'implantations dans un langage informatique quelconque.

Le poste linguistique contient:

- **un module de découpage de textes (chapitre 7)**, chargé de reconnaître sa structure en titres, paragraphes, d'identifier les mots composés ou simples, de corriger certaines erreurs orthographiques
- **le formalisme AVAG de développement de grammaires d'unification (chapitre 6)**. Ce formalisme syntaxique s'inspire de PATR-II. Nous l'avons étendu pour couvrir l'aspect sémantique. Il permet de représenter certains quantificateurs, de traduire logiquement les relations entre constituants d'une phrase ou entre phrases. Cette extension se réfère à la Théorie de la Représentation du Discours. L'opération d'unification traduit les contraintes syntaxiques et sémantiques. Notre implantation utilise directement l'unification du langage Prolog. Toutefois, Prolog ne prenant pas en compte les mécanismes d'héritage, une opération d'unification étendue gérant les contraintes hiérarchiques de types entre éléments du formalisme a été définie. Le formalisme est compilé en programme PROLOG par souci d'efficacité.
- **un couplage entre AVAG et le langage de représentation des connaissances BACK (chapitre 6)** dans lequel sont décrites une partie des connaissances du domaine. Les types AVAG correspondent à des concepts de BACK. Les contraintes hiérarchiques entre concepts sont utilisés par l'analyseur HANNA.
- **l'analyseur Hanna (chapitre 7)** qui prend en charge l'analyse d'un texte préalablement découpé et utilise grammaire et lexique décrits avec AVAG. La stratégie d'analyse est implantée à ce niveau. Diverses aides en cas d'échec de l'analyse sont possibles.
- un programme de **résolution des anaphores (chapitre 7)** conçu de façon modulaire.avec une architecture en "tableau noir".

Pour la partie **traduction des textes en règles de logique pour un système expert (chapitre 8)**, vérification de cohérence, possibilités de déduction, nous avons choisi de représenter leur contenu sous forme de règles de *logique des défauts*.

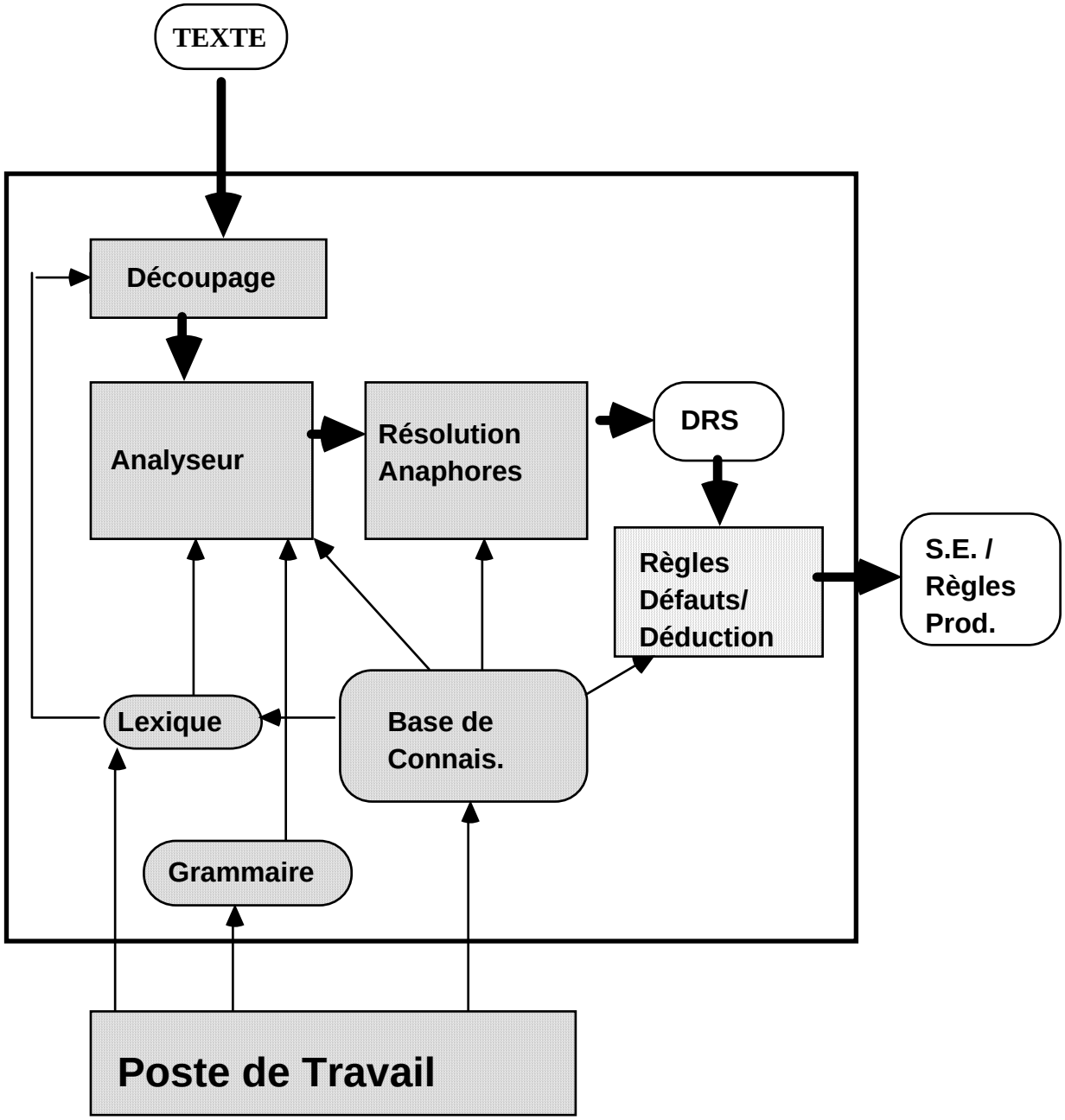


Fig 5.2: Architecture du système ACTES

5.3. SCÉNARIO D'UTILISATION

La démarche présentée ci-dessous se propose de fournir un cheminement logique dans l'utilisation du poste de travail, de la phase conception des lexique, grammaire, base de connaissances jusqu'à la génération de la représentation sémantique intermédiaire d'un texte après résolution des anaphores.

Les menus évoqués ici sont ceux qui se présentent sur l'écran graphique de l'utilisateur dès son entrée dans l'environnement ACTES.

Etape 1 : Lexique et Grammaire et Modélisation du domaine.

- Analyse du corpus et rédaction des fichiers lexique et grammaire correspondants sous le menu *EDIT*. Ces fichiers sources doivent être écrits dans le formalisme *A.V.A.G.*.
- Modélisation du domaine dans le langage de représentation des connaissances (menu *RC*). Mise en correspondance des noms des concepts avec les types sémantiques utilisés dans le lexique et la grammaire. Le classificateur de BACK ordonne automatiquement les concepts d'après leur description après vérification. A la fin de la modélisation, appel du programme de génération automatique des descriptions partielles des objets du domaine pour l'analyseur HANNA.

Etape 2 : La compilation du lexique et de la grammaire.

- Compilation du lexique source et de la grammaire source en vue de l'obtention d'un lexique et d'une grammaire exécutables en Prolog. En cas d'erreur de compilation, cette étape doit être renouvelée après correction des erreurs dans les fichiers sources. l'environnement aide aux dépistages d'erreurs.
- Après obtention des fichiers exécutables, il faut lancer la génération automatique du lexique pour le découpage du texte. Ce fichier lexique n'est pas le même que le lexique exécutable , et il est indispensable lors du découpage du texte.

La compilation des fichiers sources et la génération du lexique pour le découpage sont réalisées sous le menu *AVAG*.

Etape 3 : Le découpage du texte.

Le découpage du texte utilise en entrée le lexique généré par AVAG et le texte que l'on désire voir découpé. Il donne en sortie le texte découpé et met en évidence la structure du texte initial. Certaines erreurs typographiques sont corrigées.

Ce programme utilise de nombreux paramètres que l'utilisateur doit d'abord mettre à jour. Des explications en ligne sont fournies sur chacun d'eux.

Ce découpage est réalisé sous le menu *AC-DEC*.

Etape 4 : Analyse syntaxico-sémantique du texte, Résolution des référents.

L'analyseur (menu *HANNA*) fournit un environnement de tests nécessaire à l'analyse du texte.

- Cette étape permet d'obtenir à partir des fichiers lexique et grammaire exécutables, des connaissances du domaine et du texte découpé les différents arbres syntaxiques possibles (phrase à phrase) ainsi que les DRS associées. Ces arbres peuvent être consultés à l'écran ou écrits dans un fichier, il en est de même pour les DRS.
- En cas d'échec dans l'analyse, diverses procédures d'aides sont proposées.
- Après l'analyse, les anaphores du texte sont résolues. L'utilisateur est sollicité pour trancher les cas ambigus.

Remarque

Il est bien sûr possible de tester un lexique et une grammaire à l'aide de phrases isolées entrées directement au clavier. L'étape 3 est alors inutile. D'autre part les phases d'analyses syntaxique et sémantique peuvent être distinguées. L'utilisateur doit prévoir des lexiques et grammaires séparées en ce sens.

6. LE FORMALISME AVAG

6.1. INTRODUCTION

6.1.1. Vue d'ensemble du formalisme

Pour les **aspects syntaxiques (partie 2)** le formalisme AVAG (Attribute VAlue Grammar) s'inspire du formalisme PATR-II [SHIE 83] et plus généralement des *grammaires d'unification*.

Les grammaires d'unification permettent d'associer à chaque catégorie grammaticale une structure de traits, représentable sous forme d'arbre, ou plus précisément de graphe sans circuit (DAG). Les contraintes syntaxiques s'expriment donc en termes de contraintes sur ces DAGs et sont opérées par *unification*. AVAG apporte quelques extensions ou variantes au formalisme PATR-II, développé au SRI International. Mais surtout une grammaire écrite dans ce langage est compilée en un programme PROLOG. L'unification, opération primitive de PROLOG, est donc utilisée directement. De même, *des termes PROLOG correspondent aux structures de traits*. Il s'ensuit ainsi un gain d'efficacité par rapport aux autres implantations du type PATR-II.

Pour les aspects sémantiques (partie 3), AVAG s'inspire de la Théorie de la représentation du discours (DRT) de Kamp. Un texte est représenté comme un ensemble de structures de représentation du discours (DRS). Une DRS est implantée sous forme de structure de traits. Elle est construite à l'aide de l'opération d'unification et d'une nouvelle opération *l'augmentation*. Ainsi syntaxe et sémantique sont traitées de façon uniforme.

Pour représenter **les connaissances du domaine (partie 4)**, le système ACTES a été couplé avec le langage de représentation des connaissances BACK [LUCK 86]. Nous présentons les caractéristiques essentielles de ce langage hybride en le comparant aux représentations classiques des réseaux sémantiques et des frames. La description des connaissances du domaine peut être partiellement utilisée lors de l'analyse. L'introduction de *variables typées* et d'une *unification étendue* sur les types permettent de relier valeurs des traits de AVAG et concepts du réseau et d'appliquer les contraintes de subsomption entre ces concepts.

6.1.2. Pourquoi compiler le formalisme AVAG ?

Rapprocher structures de traits et structures de données PROLOG, d'une part, unification de la théorie des grammaires de traits et unification PROLOG, d'autre part, pose deux problèmes:

- la notion d'**ordre de description des trait**s est indifférente dans la théorie, pas en PROLOG. Par exemple, si on représente une structure de traits sous forme de liste à plusieurs niveaux, ces deux listes ne peuvent s'unifier en PROLOG:

```
(1)  [ accord,[genre, feminin],[nombre, singulier]]
     [ accord,[nombre, singulier],[genre, feminin]]
```

- deux structures de traits peuvent s'unifier, même si elles n'ont pas le même **nombre d'arguments**, ce qui n'est pas le cas en PROLOG. Ainsi ces deux listes ne s'unifient pas:

```
(2)  [ accord,[genre, feminin],[nombre, singulier]]
     [ accord,[genre, feminin]]
```

Ces différences expliquent le fait que nous ayons dû concevoir une compilation à deux passes: la première recense et ordonne les traits (de façon transparente à l'utilisateur); la deuxième, à partir de chaque entrée lexicale et de chaque règle AVAG, et des structures précompilées dans la première passe, génère les clauses PROLOG correspondantes. Le résultat est une grammaire de type DCG. Signalons en outre, que cette précompilation des traits permet de ne pas stocker dans les structures de traits (les listes PROLOG) les noms des traits, mais seulement leur valeur. Elle autorise aussi une description partielle des traits.

6.1.3. Descriptions syntaxique et sémantique séparées ?

Dans les paragraphes suivants du chapitre, nous distinguons les aspects syntaxiques et sémantiques du formalisme AVAG. Cette présentation séparée ne se justifie que pour des raisons didactiques. Les contraintes syntaxiques et sémantiques peuvent être décrites simultanément ou séparément, au gré de l'utilisateur. Toutes ces contraintes s'expriment dans les structures de traits.

Si les aspects syntaxiques et sémantiques sont séparées, l'analyse comprendra d'abord une phase syntaxique qui, à partir d'un énoncé et d'une grammaire syntaxique produira un **arbre syntaxique** représentant la structure de surface de l'énoncé. Cet arbre servira d'entrée à la grammaire sémantique qui produira la représentation sémantique sous forme d'une structure de traits.(cf figure 6.1) Grammaires syntaxique et sémantique sont écrites dans le même formalisme AVAG. L'arbre syntaxique est soit l'**arbre système** généré automatiquement par le système, soit l'**arbre utilisateur** décrit explicitement dans la grammaire (cf infra).

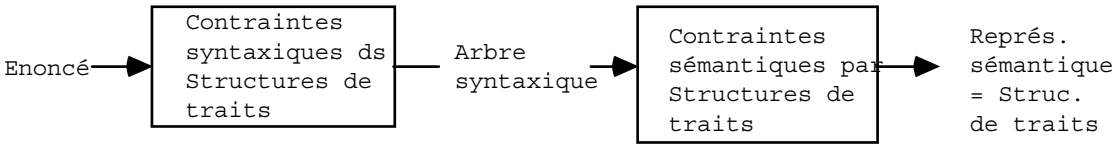


Fig 6.1: Analyses syntaxique et sémantique séparées

En cas d'analyse syntaxico-sémantique, une seule grammaire suffit. Elle n'utilise que les structures de traits (cf figure 6.2). L'arbre syntaxique n'est utilisé que lors de la recherche de référents.

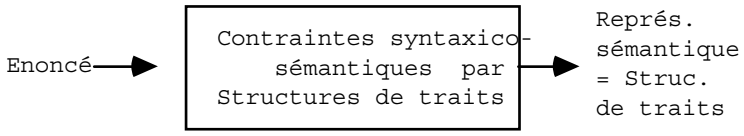


Fig 6.2: Une seule phase d'analyse

6.2. ASPECTS SYNTAXIQUES DU FORMALISME

Le formalisme syntaxique AVAG est conçu comme un outil linguistique proche de PATR-II et peut être considéré comme une formulation opérationnelle de GPSG. Le compilateur associé traduit toute grammaire écrite dans le formalisme AVAG en une grammaire DCG interprétable en PROLOG.

Nous présenterons quelques aspects du formalisme en mettant l'accent sur les opérations sur les traits, unification et autres opérations, sur les problèmes d'instanciation des traits, sur l'expression de constituants non obligatoires. En outre, quelques indications succinctes de compilation sont données (pour une description détaillée cf [CHAN 86]).

L'essentiel des informations syntaxiques est contenu dans le lexique. Les relations entre constituants grammaticaux décrites dans les règles, dépendent de ces informations. La grammaire décrite dans ce paragraphe permet d'analyser la phrase:

```
(3)     on utilise la vitesse-sol
```

Outre les contraintes d'accord, la grammaire permet d'exprimer les contraintes de type sur la nature des informations. Le verbe "utiliser" ne porte que sur des informations de type "entrée" ou "paramètre". La "vitesse-sol" est bien une information de type "paramètre".

6.2.1. Les entrées lexicales

Voici une façon simplifiée de décrire l'entrée lexicale du mot "utilise":

```
(4)   utilise:
< cat > = verbe                (i)
<accord nombre>= singulier     (ii)
<accord personne> = 3          (iii)
< type >=  {ENTREE,PARAM}      (iv)
```

Les quatre équations expriment des contraintes sur les traits du mot "utilise". L'ordre des contraintes est indifférent. Le symbole "=" est l' opérateur d'unification . Ces contraintes sont ici des instanciations des traits à des valeurs atomiques:

- dans (ii) le trait 'nombre' a la valeur 'singulier'. 'accord nombre' est le *chemin* définissant le trait.

- dans (iv) la valeur du trait 'type' est aussi atomique. La notation entre accolades exprime la contrainte posée sur la valeur du trait. Cette valeur sera déterminée ultérieurement lors de l'analyse et sera égale à 'ENTREE', ou à 'PARAM'. On pourrait encore noter (iv): <*type*> = *X* , avec X variable typée, X □ {ENTREE,PARAM}. Cette contrainte à action différée s'exprime en PROLOG avec la primitive *freeze*.

Il existe des facilités syntaxiques du formalisme AVAG permettant de décrire des ensembles d'équations une seule fois, et d'y faire ensuite référence, ce sont les *gabarits* .

Voici celui correspondant à l'accord féminin singulier:

```
(5)   soit femsing:
<accord genre> = feminin
<accord nombre> = singulier
```

On fait référence à ce gabarit ainsi:

```
(6)   vitesse-sol:
<cat> = nominfo
femsing
<type> = PARAM.
```

Il est possible, pour décrire une entrée lexicale, de *faire référence à un mot du lexique* décrit précédemment en ajoutant ou en enlevant une partie de sa description:

```
(7)   position-commande-trains:
^vitesse-sol
<type> = BOOLEEN.
```

"position-commande-trains" se décrit comme "vitesse-sol", mais son trait "type" a la valeur "booléen".

Les autres entrées lexicales peuvent se définir de la façon suivante:

```
(8)   la:
<cat> = art
femsing
<type> = def.
```

```
(9)   on:
<cat> = pronom
<accord nombre>= singulier
<accord personne> = 3
<type> = indef.
```

Sur l'exemple du mot "la", expliquons le principe de la compilation d'une entrée lexicale . Lors de la première passe de compilation, tous les traits associés à la catégorie 'art' sont recensés et ordonnés. Une structure ne comprenant que les noms des traits est produite. Soit, par exemple :

```
(10)   [ cat,effacer type, accord,[genre,nombre,personne] ]
```

La présence du trait 'effacer' sera justifiée plus loin.

La deuxième passe de compilation produit le fait PROLOG suivant, à partir de (8) et (10):

```
(11)  la( [ art,X2, def, [ feminin,singulier, X1 ]]).
```

L'argument du prédicat 'la' est la structure de traits associée au mot (seules figurent ici les valeurs des traits).

6.2.2. Les règles

Une règle AVAG est composée d'une règle de réécriture hors-contexte et d'un ensemble (éventuellement vide) d'équations. Voici, par exemple, la règle d'un groupe nominal simple:

```
(12)  gn -> art nom                    (i)
      <art accord> = <nom accord>      (ii)
      <gn head accord> = <nom accord>.   (iii)
```

- (ii) exprime la contrainte d'accord entre l'article et le nom.

- (iii) permet de remonter (percoler) tous les traits placés dans 'nom' sous 'accord' dans le trait 'accord' du groupe nominal 'gn'.

Nous n'expliciterons pas, par souci de lisibilité, le détail de la forme compilée de cette règle . C'est une règle DCG de forme générale :

```
(13)   gn(AU1, gn(AS2,AS3), L1, X1, X3) :-
           art(AU2, AS2, L2, X1, X2),
           nom(AU3, AS3, L3, X2, X3).
```

AU1, *AU2*, *AU3*, sont les arbres utilisateurs (cf infra). *gn(AS2,AS3)*, *AS2*, *AS3*, sont les arbres syntaxiques (structure de surface du groupe de mots analysé). *X1*, *X2*, *X3* sont les listes d'entrée et de sortie contenant les mots à analyser.

L1, L2, L3 sont les structures de traits (non détaillées ici) correspondant à chaque catégorie. Elles sont construites à partir des descriptions des structures recensées, pour chaque catégorie, pendant la première passe de compilation (cf (10)). Dans chacune d'elles, seules les valeurs des traits sont présentes. Les contraintes exprimées dans les équations de (12) sont représentées par des contraintes d'unification PROLOG (même nom de variable pour des traits qui s'unifient).

6.2.3. La négation

Dans les équations ci-dessus, les contraintes sur les traits sont réalisées par l'opération d'unification. Mais il peut être intéressant de décrire des contraintes stipulant que la valeur d'un trait doit être différente de celle d'un autre. L' opération de négation (notée "#") permet d'exprimer cela.

La gestion de cette contrainte est particulière. Elle modifie l'opération d'unification sur les structures de traits qui contiennent les deux traits sur lesquels portent la négation. Elle doit, de plus, être propagée lors des unifications qui réussissent. Une telle contrainte correspond à l'effet de la primitive 'dif' de PROLOG II.

Par exemple, dans une règle décrivant un groupe verbal, on peut dire que la règle ne doit pas s'appliquer à des verbes intransitifs (donc n'admettant pas d'objet direct), plutôt que d'énumérer toutes les sous-catégories de verbes auxquels elle peut s'appliquer.

```
(14) gv -> verb gn
<gv head accord> = <verb head accord>
<gv head > = <verb head>
< verb souscat> #  intrans.
```

gv est le groupe verbal

6.2.4. Précédence locale

Dans le formalisme GPSG, il y a d'un côté des règles de dominance immédiate (ID) et de l'autre des précédences linéaires (LP) qui s'appliquent à toute la grammaire. Nous avons réduit la portée des précédences. Elles s'appliquent à la règle dans laquelle elles sont définies (on peut parler de précédences locales).

La règle (14) peut-être vue comme une règle de dominance immédiate (noté ID dans GPSG) dans laquelle la précédence linéaire est implicite .

Donnons un exemple dans lequel la précédence est explicite. Un constituant groupe verbal 'gv' peut être décrit ainsi:

```
(15) gv ->  verb , gn , gp   [ verb<gn  verb<gp ]
<verb head souscat> = 3
<verb head pform> = <gp pform>
<gv head> = <verb head>.
```

gp est le groupe prépositionnel. Les virgules entre les catégories indiquent que ces constituants peuvent être placés dans un ordre quelconque (toutes les permutations de gv, gn et gp) dans la règle de réécriture. Les "inéquations" entre crochets limitent ces permutations et se lisent : un verbe (verb) est situé avant un groupe nominal (gn) et avant un groupe prépositionnel (gp).

(15) est donc équivalent à (16) et (17):

```
(16) gv -> verb gn gp
<verb head souscat> = 3
<verb head pform> = <gp pform>
<gv head> = <verb head>.

(17) gv -> verb gp gn
<verb head souscat> = 3
<verb head pform> = <gp pform>
<gv head> = <verb head>.
```

6.2.5. Eléments facultatifs, vides

6.2.5.1. Eléments facultatifs

Dans AVAG il est possible d'exprimer le fait que certains constituants de la grammaire sont facultatifs ou vides. Des problèmes de contrôle se posent alors, notamment liés à des traits non instanciés.

Les constituants non obligatoires d'une règle sont spécifiés dans GPSG par l'utilisation d'une méta-règle. AVAG procure directement cette méta-règle par le biais d'un méta-symbole "/". Ainsi si nous appelons 'ajout_gche' l'ajout à gauche du nom (comprenant en particulier article/cardinal ...), la règle ci-dessous décrit qu'un groupe nominal est constitué d'un ajout à gauche et d'un nom, ou bien simplement d'un nom:

```
(18) gn -> /ajout_gche nom
<ajout_gche accord> = <nom accord>  (i)
...
```

Il est intéressant de pouvoir contrôler l'effacement de constituants facultatifs dans une règle, soit pour ne pas autoriser leur effacement simultané, soit parce que lorsqu'un constituant est effacé, une équation telle que (i) n'exprime plus aucune contrainte (structure de traits variable). Pour cela, l'utilisateur dispose du trait prédéfini 'effacer' présent dans chaque structure (cf.exemple 10) et positionné à des valeurs différentes suivant que l'effacement a lieu ou non lors de l'analyse.

6.2.5.2. Eléments vides

La notion d'élément facultatif est différente de celle d'élément qui, bien que pouvant prendre la valeur vide, est indispensable.

Soit, par exemple, le groupe nominal comprenant une relative:

(19) la condition qui porte sur la vitesse

La relative (en italique souligné) peut s'analyser comme une phrase dont le groupe nominal sujet (indispensable) se réécrit en symbole vide . Pour exprimer la contrainte d'accord entre le sujet (ici vide) et le verbe, on peut utiliser un trait (que nous appelerons 'slash' dans l'exemple, en référence au trait SLASH de GPSG). Ce trait transportera les valeurs des traits 'genre' et 'nombre' du groupe nominal "la condition" dans la relative afin de vérifier l'accord.

Un extrait de grammaire correspondant à cette description pourrait s'écrire ainsi :

```
(20) gn.-> art nom [qui] ph          (i)
     <art accord> = <nom accord>     (ii)
     ph slash> = <nom accord>        (iii)
     <gn accord> = <nom accord>.


(21) ph  ->  gn  gv                  (i)
     <ph slash> = <gn accord>        (ii)
     <gn accord> = <gv accord>.      (iii)


(22) gn ->                           (i)
     <gn accord> # X0.               (ii)
```

(20) permet de reconnaître le groupe de mots (19). Les crochets autour du pronom relatif "qui" indiquent que ce mot pourra être reconnu lors de l'analyse sans avoir d'entrée lexicale (*mot non défini*). Les traits 'genre' et 'nombre' du nom sont transmis dans le trait 'slash' de la phrase de nom 'ph' (équation (iii)).

(21) indique qu'une phrase est composée d'un groupe nominal sujet ('gn') et d'un groupe verbal ('gv'). Le groupe verbal reconnaît "transporte du vin". (i) transmet les traits du nom contenu dans 'slash' dans celui du trait 'accord' du groupe nominal vide. (ii) exprime l'accord entre le groupe nominal et le groupe verbal.

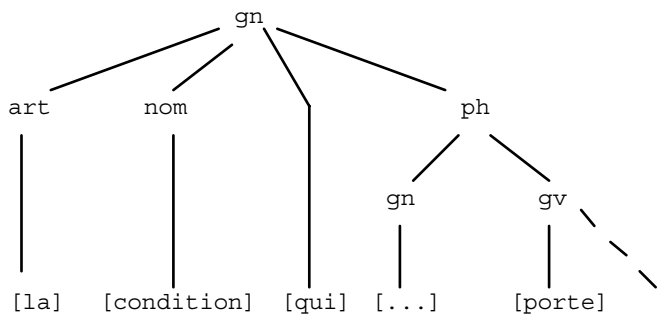
(22) est la règle de réécriture du groupe nominal vide. Il est nécessaire de pouvoir contrôler les circonstances dans lesquelles un élément peut être vide. Sinon la grammaire accepterait une phrase sans sujet comme correcte. Cela peut se faire à l'aide du trait 'effacer' positionné à une valeur différente du cas précédent ou encore, en indiquant que le trait 'slash' ne peut être variable (équation (ii)). L'équation (ii) garantit que la règle ne s'appliquera que dans le cas d'une relative.

6.2.6. Arbre utilisateur et arbre système

Le résultat de l'analyse syntaxique est un arbre syntaxique. AVAG ajoute automatiquement au cours de la compilation les arguments nécessaires à l'obtention d'un tel arbre (que nous appelons l'arbre système). Il montre la structure de surface de la phrase analysée.

L'arbre système correspondant à l'analyse du groupe nominal de l'exemple (19) sera:

(23)



Mais l'utilisateur a la possibilité de décrire explicitement la façon dont l'arbre doit être construit en ajoutant les arguments nécessaire aux catégories grammaticales de chaque règle de réécriture.

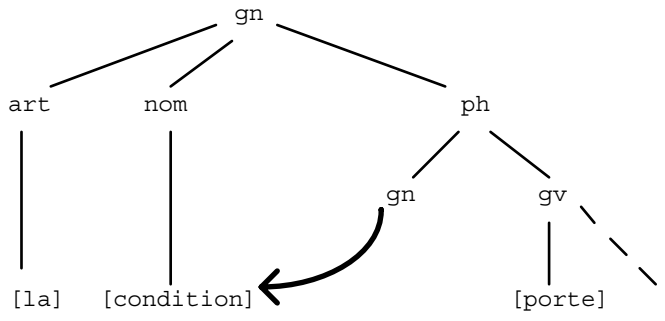
Pour faire apparaître, par exemple le sujet "condition" dans la relative, on peut modifier ainsi la règle du groupe nominal donné en (20):

```
(24) gn(gn(X1,X2,ajout(X3))) -> art(X1) nom(X2) [qui] ph(X3)
<art accord> = <nom accord>
<ph slash head> = <nom accord>
<ph slash arg> = X2            (i)
<gn accord> = <nom accord>.
```

L'équation (i) permet de passer l'arbre syntaxique venant du 'nom' au 'ph' sous le trait 'slash arg'.

En modifiant de la même façon les autres règle ('ph', 'gv', 'nom' ...), on peut obtenir l'arbre utilisateur suivant pour la relative:

(25)



6.2.7. Coordination et structures de traits

Dans le système ACTES, le traitement de la conjonction de coordination (du moins de certains cas de coordination sans ellipse) est pris en charge au niveau de l'analyseur et non directement dans la grammaire [SEDO 85]. Avant la compilation de sa grammaire, l'utilisateur doit spécifier les catégories sur lesquelles il autorise la coordination.

Si, par exemple, l'utilisateur désire analyser des phrases comme (26), il indique que deux groupe nominaux (*gn*) peuvent être conjoints.

```
(26 a)    le train et la trappe sont bloqués
(26 b)    les train et trappe sont bloqués
...
```

Toutes les règles comprenant un *gn* dans leur partie droite seront alors transformées lors de la compilation. Ainsi (26 c) donnera (26 d):

```
(26 c)   ph -> gn gv

(26 d)   ph(L1) :- conj(gn(L2))  gv(L2)
```

et *conj* se définit ainsi (en omettant les arguments sur les listes d'entrée et de sortie):

```
conj(P) :-
     call(P),
     rest_conj(P).

rest_conj(P):-
     call(P),
     conjonction(P), /* reconnaissance du mot ou
                        signe de ponctuation
                      indiquant la conjonction de coordination */
     !,
     rest_conj(P).
rest_conj(P).
```

Comment traiter alors le problème des accords entre deux structures de traits conjointes? Supposons que les structures de traits correspondantes à "train" et trappe" soient respectivement:

```
(27)
cat : GN             cat: GN
acc : genre : MAS    acc: genre : FEM
      nbre  : SING         nbre  : SING
```

Le résultat de la conjonction devrait donné:

```
(28)  cat : GN
acc : genre : MAS
       nbre : PLU
```

Lauri Karttunen [KART 84] propose d'opérer une *généralisation* sur les deux structures de départ. La généralisation de deux structures est la structure la plus générale subsumée par les deux premières; ou encore le sous-graphe maximal de deux graphes. Mais que faire lorsque deux traits ont des valeurs conflictuelles ?

Pour certains cas, on pourrait envisager d'introduire une notion de hiérarchie entre les valeurs atomiques (MAS">"FEM). Mais sans extension de l'unification, cette façon de procéder est contradictoire avec l'unification présentée ici. Karttunen, critiquant cette approche, suggère d'utiliser l'opération de négation sur les traits; mais la formulation des contraintes d'accord n'est pas très aisée.

Dans notre système, plus simplement, nous avons étendu le formalisme syntaxique AVAG pour permettre à l'utilisateur d'expliciter (en dehors de sa grammaire), pour chaque trait d'une catégorie, quel doit être le résultat de la coordination sur les valeurs possibles du trait.

Voici comment on peut décrire les contraintes dûes à la coordination sur les traits genre et nombre d'un groupe nominal comme dans les exemples (27) et (28):

(29)

```
 1    <cat>= GN
<coord acc genre> = FEM
<coord acc genre> = FEM
<result acc genre> = FEM.

2     <cat>= GN
<coord acc genre> = X0
<coord acc genre> = MAS
<result acc genre> = MAS.

3     <cat>= GN
<coord acc genre> = MAS
<coord acc genre> = X0
<result acc genre> = MAS.

4     <cat>= GN
<coord acc nbre> = X0
<coord acc nbre> = X1
<result acc nbre> = PLU.
```

(2) se lit: le résultat de la coordination d'un groupe nominal dont la valeur du trait 'genre' est indéfini avec un groupe nominal dont la valeur du trait 'genre' est MAS sera un groupe nominal ayant un trait 'genre' de valeur MAS. On remarquera qu'il est nécessaire de décrire le cas inverse en (3). Les contraintes de coordination ne sont pas commutatives.

Cette description est ensuite compilée par AVAG en suivant le même codage des structures de traits que celui de la grammaire. Si la structure de traits correspondant au *gn* est (30), alors le compilateur générera les structures (31) correspondantes à (29).

(30) `[ cat, accord,[genre,nombre] ]`

```
(31)
conj_feature(gn, [gn, [fem,_]], [gn, [fem,_]], [gn, [fem,plu]]):-!.
conj_feature(gn, [gn, [_,_]], [gn, [mas,_]], [gn, [mas,plu]]):-!.
conj_feature(gn, [gn, [mas,_]], [gn, [_,_]], [gn, [mas,plu]]):-!.
```

Lors de l'analyse, le programme qui gère la coordination accède à la description compilée et produit les résultats désirés. Voici la nouvelle définition, simplifiée, de *conj*. La description complète peut être trouvée dans l'annexe.

```
(32)

conj(Q) :-
   Q =.. [P,Traits],
   Q1 =..[P,Traits1],
   call(Q1),
   Q2 =..[P,Traits1],
   rest_conj(Q2,Traits).

rest_conj(Q,T) :-
   Q =..[P,T1],
   conjonction(Q),
   Q2 =..[P,T2],
   call(Q2),
   generaliser(P,T1,T2,T3),
   !,
   Q3 =..[P,T3,],
   rest_conj(Q3,T).

rest_conj(Q,S,A,T).


generaliser(Cat,[Cat|L1],[Cat|L2],[Cat|L3]) :-
   conj_feature(Cat,[Cat|L1],[Cat|L2],[Cat|L3]),
   !.

generaliser(_,_,_,_):-
  traitement des erreurs de coordination
   .....
```

Cette façon de faire est certes statique au sens où, pour deux structures à conjoindre, tous les résultats sont calculés à l'avance. Mais la compilation permet d'accomplir ce travail de façon transparente à l'utilisateur, et le résultat, lors de l'exécution, est efficace.

6.2.8. Autres facilités syntaxiques

6.2.8.1. Catégorie indicée

Dans une règle, une même catégorie peut figurer plusieurs fois. Pour distinguer les différentes occurrences de cette catégorie , il suffit de les indicer (sauf la première occurrence) par la caractère '$' suivi du numéro d'occurrence.

```
(33) gv ->  verb gn gp gp$1
```

Dans cet exemple un groupe verbal est composé d'un verbe, d'un groupe nominal, d'un premier groupe prépositionnel suivi d'un deuxième (qui peut être vide).

6.2.8.2. Référence à un mot du lexique dans une catégorie

Si on désire décrire, dans une règle, une construction qui ne s'appliquera que pour un certain mot du lexique, il suffit de faire référence à un mot du lexique dans une catégorie.

```
(34) gp -> prep(*pour) gv gn
<gv temps> = infinitif
...
```

La règle ne s'applique que si la préposition commence par le mot "pour", mot qui doit être défini dans le lexique.

6.2.8.3. Terme Prolog

Bien que l'approche choisie dans AVAG soit déclarative et essaye de proposer un formalisme permettant de développer des grammaires en se préoccupant le moins possible de problèmes d'implantation, il peut toutefois être nécessaire de faire appel explicitement à des prédicats PROLOG.

Ainsi il est possible de faire appel à un terme PROLOG; en l'entourant à l'aide des accolades '{' et '}'. Si les variables utilisées dans ce terme correspondent à la syntaxe des variables AVAG, l'unification se fera normalement avec les arguments des catégories de la règle de réécriture ou les équations.

6.3. ASPECTS SÉMANTIQUES ET DRT

6.3.1. Implanter les DRS dans des structures de traits

La construction et la représentation des DRS associées aux parties de discours à analyser, s'effectuent à partir d'opérations sur des données sémantiques, correspondant à une manipulation de structures de traits. Ces données sémantiques sont paramétrables; pour les besoins de la maquette nous avons donné à ces traits un nom arbitraire.

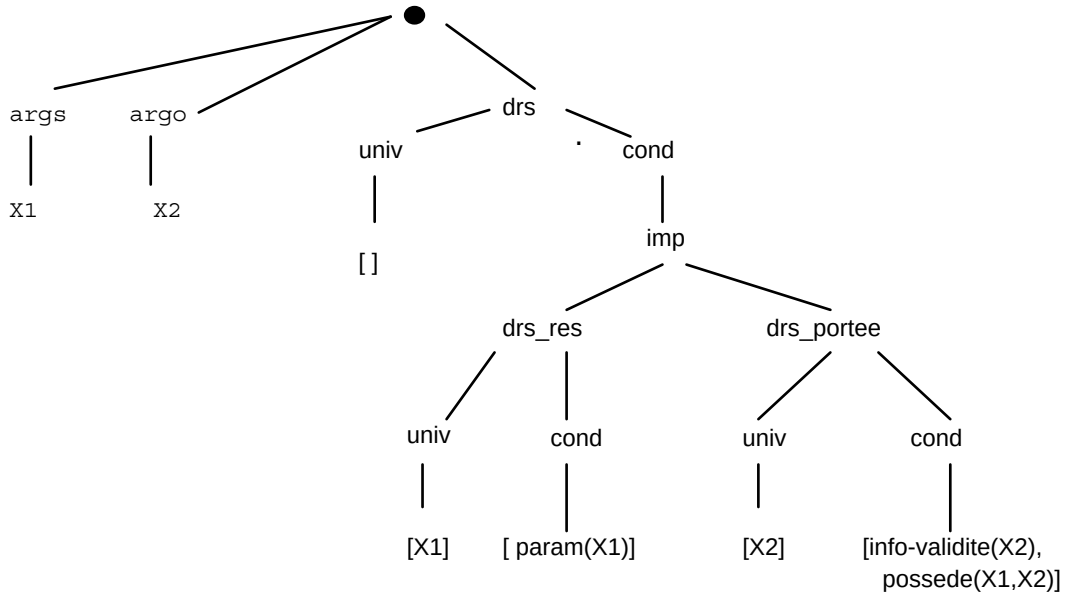
Dans la phrase "*les paramètres qui possèdent une info de validité sont conditionnés par elle*", la 'drs_res' (ou 'drs_ante') correspond à la première partie de la phrase (non soulignée). La 'drs_portée' (ou 'drs_cons'), elle, représente la portée du quantificateur, soit *"sont conditionnés par elle"* (Pour plus de précisions, voir [KLEI 86]).

La DRS associée à la phrase:

```
(35) Tout paramètre possède une info de validité
```

peut donc être représentée par le DAG (cf la figure 3.6 pour le dessin de la DRS correspondante):

(36)



où la 'drs principale' est composée de la 'drs_res' et de la 'drs_portée', ceci grâce au quantificateur "*tout*".

Dans le cas des quantificateurs "tout", "chaque", "tous les" etc..., les DRS partielles formant une condition complexe de la DRS principale, appelée DRS implicative, devront être indicées, pour référencer la DRS dans laquelle on va éventuellement continuer l'analyse du discours.

Les manipulations sur les structures de traits sont exprimées à l'aide de l'unification, comme pour l'unification des structures syntaxiques précédemment décrites. Ainsi, l'on peut unifier les traits 'args' et 'argo' avec les variables 'X1' et 'X2', le trait 'drs' avec une 'drs_res' ou une 'drs_portée', etc...

Cependant, cette opération d'unification est insuffisante car elle ne permet pas l'augmentation d'une même structure par plusieurs composants. En effet, les structures 'Univers' et 'Cond' se présentent comme des listes ouvertes, susceptibles d'être augmentées autant qu'il est nécessaire pendant l'analyse par de nouveaux référents et de nouvelles conditions. Pour pouvoir inclure ceux-ci, nous utilisons donc une autre opération: *l'augmentation* (notée " =:: ").

6.3.2. L'opération d'augmentation et l'unification

Quelle différence y a-t-il entre l'opération d'augmentation et celle d'unification? Toutes deux opèrent sur des graphes sans circuit, mais l'augmentation opère différemment sur les feuilles.

Dans la partie syntaxique concernant le formalisme AVAG, les valeurs des feuilles d'une structure de traits ne pouvaient être qu'atomiques. Maintenant, les valeurs des feuilles seront de deux types: atome ou liste. Une *liste* est un ensemble ordonné d'atomes sans répétition[1] .Elle est notée entre crochets ("[", "]").

Précisons les notations utilisées dans les équations:

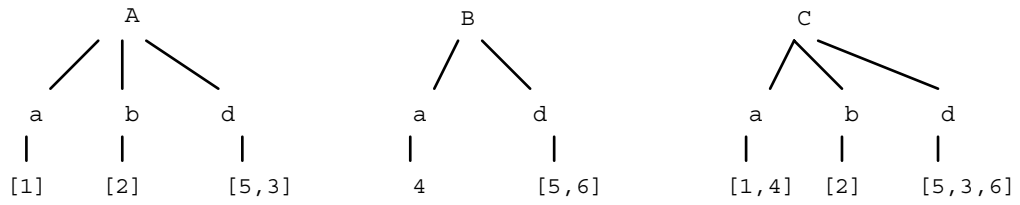
(37)

```
<a> =   1          (i)
<a> =:: 1          (ii)
<a> =  [1]         (iii)
<a> =:: 2          (iv)
<a> = [1, 2]       (v)
```

(i): le trait 'a' a pour valeur l'atome '1'. Le symbole "=" représente l'opération d'unification.
(ii): la valeur du trait 'a' est augmentée de l'atome '1'. Si ce trait n'était pas instancié auparavant, sa valeur devient la liste [1].
(iii) : La valeur du trait 'a' s'unifie avec la liste [1]. Donc même résultat que (ii).
(v): l'ordre des opérations d'augmentation est important. Ainsi (v) est le résultat de l'application de (ii) et (iv), et non (iv) et (ii).

Nous ne détaillerons pas ici la façon dont l'unification et l'augmentation opèrent sur des feuilles de même type ou de types différents. Donnons seulement l'exemple de l'augmentation de deux structures A et B. 'Augmenter A par B' (noté 'A =:: B') modifie A de la façon suivante:

(38)



Dans cet exemple, on voit que l'augmentation procède de la même façon que l'unification sur les noeuds non terminaux.

[1] Le choix d'ordonner les atomes permet dans la liste correspondant à la valeur du trait 'univers' d'une DRS de faire une recherche des référents dans l'ordre de leur apparition.

6.3.3. Exemple de lexique et de grammaire sémantique

Avant de traiter les règles grammaticales, il convient de décrire la composition des entrées lexicales, à partir desquelles sont construites les structures de base, et en particulier des quantificateurs puisque ce sont eux qui déterminent la structure de la DRS principale. Ainsi, le traitement de la phrase (35), s'effectue à partir des équations que nous allons décrire ci-dessous.

6.3.3.1. Les entrées lexicales

Les variables utilisées dans les contraintes suivantes sont les variables typées 'X' pour les individus, 'K1' et 'K2' pour les DRS.

```
(39)    tout:
        < cat >  = < art >            (i)
        < arg >  =   X                (ii)
        <drs cond>=:: imp(K1,K2)      (iii)
        <drs_res> = K1                (iv)
        <drs_portée>= K2              (v)
        <drs_res univ>=:: X           (vi)
```

(iii) indique que le quantificateur "tout" introduit la condition complexe "imp(K1,K2)" dans la liste des conditions de la DRS globale. Cette condition est l'implication de deux sous-DRS dont la première, K1, est la restriction du quantificateur, et la seconde, K2, en est la portée ((iv) et (v)) .

```
(40)       paramètre:
           < cat > = nom
           < arg > = X
           <pred > = paramètre(X)
      possède:
           < cat > = verbe
           <args > = X1
           <argo > = X2
           <pred > = posséder(X1,X2)
```

6.3.3.2. Les règles grammaticales

Les règles qui suivent expriment l'assignation d'une DRS ou d'une DRS partielle à un groupe verbal, un groupe nominal ou une phrase:

```
(41)   gv -> verb gn
           <gv args> = <verb args>       (i)
           <gv argo> = <verb argo>       (ii)
           <gn arg > = <verb argo>       (iii)
           <gn drs_portée>=::<verb pred> (iv)
           <gv drs> =:: <gn drs>         (v)
```

Le groupe verbal est composé d'un verbe suivi d'un groupe nominal (vide ou non). Son argument objet est le 'gn' qui suit le verbe ((ii) et (iii)). La portée du quantificateur de ce 'gn' va donc être augmentée du prédicat associé au verbe (iv). Enfin, (v) dit que la DRS du 'gn' va elle-même compléter celle du 'gv'.

```
(42)  gn -> art nom
      <art arg> = <nom arg>                (i)
      <gn arg > = <nom arg>                (ii)
      <gn drs_portée>=<art drs_portée>     (iii)
      <gn drs > = <art drs>                (iv)
      <art.drs_res cond>=::<nom pred>      (v)
```

C'est l'article qui va déterminer la structure de représentation du 'gn'. C'est pourquoi sa portée est celle du 'gn' (iii) et sa DRS restriction contient le prédicat associé au nom (v).

```
(43)  ph -> gn gv
           <ph drs> =:: <gn drs>       (i)
           <gn arg> =  <gv args>       (ii)
           <gn drs_portée>=<gv drs>    (iii)
```

C'est le 'gn' qui impose la structure de la phrase par l'intermédiaire de son quantificateur. La DRS de 'ph' est donc augmentée de la DRS du 'gn' (i), la portée de celui-ci étant égale à la DRS du 'gv' (iii).

6.4. REPRÉSENTATION DES CONNAISSANCES, TYPAGE, UNIFICATION ÉTENDUE

6.4.1. Connexion avec un langage de représentation des connaissances

Les objets mentionnés dans les textes que nous traitons sont les constituants d'un avion et les informations sur ces constituants. Ces objets et leurs relations sont décrits à l'aide d'un langage de représentation des connaissances. Cette somme d'informations doit être accessible lors de l'analyse syntaxico-sémantique du texte, puis lors des phases ultérieures de déduction et de construction des règles tirées des formules sémantiques (DRS).

Quelles sont les informations nécessaires pendant l'analyse syntaxico-sémantique ? Pour représenter le sens d'un mot, la notion de type est fondamentale. Une hiérarchie simple entre les types est souvent trop pauvre pour décrire les objets d'un domaine. Il faudrait disposer d'un système avec multi-héritage. On peut exprimer les contraintes de sélection entre les mots à l'aide de cas sémantiques (cf les grammaires de cas). Cela se représente bien dans le formalisme des réseaux sémantiques par les rôles, qui expriment des relations entre concepts. Parmi ces rôles, la relation entre un objet et ses parties constituantes (souvent nommée PARTIE_DE) est très utilisée particulièrement pour la résolution de référents.

Toutes ces informations se représentent bien à l'aide d'un réseau sémantique. C'est pourquoi nous avons décidé de connecter ACTES au langage BACK ([LUCK 86], [PELT 87]). BACK est un système de représentation hybride comprenant une partie terminologique (réseau sémantique, descriptions des concepts, relations taxinomiques), et une partie assertionnelle (assertions logiques sur les faits du monde reliés à des concepts). Sa sémantique est bien définie et s'inscrit dans notre approche déclarative. Des conditions matérielles ont aussi guidées notre choix (car BACK n'est pas le seul langage à satisfaire les exigences précédentes) : sa disponibilité, son implantation en C_PROLOG.

Nous ne sommes pas en mesure aujourd'hui d'exploiter toute la richesse[2] de ce langage, et la connexion BACK-ACTES est limitée pour des raisons aussi bien déclaratives (relations précises entre les deux formalismes) que procédurales (PROLOG différents, temps de réponses de BACK). Malgré ces limitations, nous ne pensions pas qu'il était nécessaire de développer un formalisme hiérarchique restreint ad hoc pour notre application.

Il est beaucoup plus intéressant, au niveau méthodologique, de décrire le plus finement possible (en se servant, par exemple, de la richesse des descripteurs de rôles) les connaissances d'un domaine, de disposer de moyens de classement automatique de ces descriptions. Même si nous n'utilisons finalement (pour l'instant) qu'une petite partie de ces descriptions, nous avons un minimum de garanties sur leur cohérence globale.

Cette façon de poser les problèmes rejoint en partie la démarche des chercheurs en représentation des connaissances engagés dans la conception des systèmes hybrides. Avant d'expliciter les connexions AVAG-BACK, nous voudrions justement rappeler les raisons de ce travail en profondeur.

6.4.2. Le formalisme BACK

BACK est un langage hybride de représentation des connaissances (RC). Il s'inspire en partie des idées rassemblées, au début des années 80, par un groupe de chercheurs en IA autour du développement du langage KL-ONE [BRAC 85a]. Leur démarche consistait à mettre à plat des notations et concepts utilisés plus ou moins informellement jusque là par les principaux courants IA en RC.

Afin d'introduire les principales caractéristiques de BACK nous survolerons les avantages et inconvénients de quelques formalismes populaires en RC. Rappelons que, pour avoir une vue détaillée de chacun d'eux et des débats évoqués, on peut se reporter à [BRAC 85b].

---

[2] Malgré sa richesse, Back est aussi limité dans ses possibilités d'expression sur des points qui nous intéressent : gestion des ensembles, réel lien PARTIE_DE,.... mais ces problèmes subsitent encore au niveau de tous les formalismes.

Cette présentation nous permettra également d'introduire la terminologie que nous utilisons dans notre système.

6.4.2.1. Base de Connaissances et langage de Représentation des connaissances

Pour différencier une base de connaissances d'une simple base de données et un formalisme de RC d'un langage de manipulation et d'interrogation, les chercheurs en IA ont coutume de citer l'hypothèse de RC (*Knowledge Representation Hypothesis*) de Smith [SMIT 82]. Un système à base de connaissances doit être tel que:

- Les structures de sa base puissent être interprétées comme des propositions représentant la totalité du savoir de la base. Ce qui implique, en particulier, que ces structures soient des expressions d'un langage muni d'une notion de valeur de vérité.
- C'est la présence de ces structures qui explique le comportement global du système.

Deux critères sont considérés comme importants pour un langage de RC: sa puissance d'expression (quelles informations peut-on exprimer dans ce langage) et son adéquation expressive (y-a-t-il une correspondance simple entre les entités du domaine de la base et la façon de les représenter dans le formalisme ?).

Afin d'éviter l'utilisation de paramètres plus ou moins empiriques pour comparer des formalismes de RC, on donne souvent une définition logique d'une base de connaissances.

Une base de connaissances est constituée d'un ensemble de formules bien formées d'une logique L (les axiomes) qui décrivent un état du domaine et d'une relation de dérivabilité qui spécifie ce qui peut être déduit des axiomes à l'aide des règles d'inférences de L. Un énoncé E appartient à une base de connaissances ssi E peut être dérivé par application des règles d'inférences de L aux axiomes propres de la théorie.

Il faut bien voir que le langage logique utilisé ici n'est pas le formalisme de RC, mais est seulement un langage pivot permettant d'étudier les propriétés du formalisme en faisant appel à des procédures bien connues de démonstration.

6.4.2.2. Logique des prédicats du premier ordre

Compte-tenu de ce qui vient d'être dit, la logique des prédicats est un bon candidat comme formalisme de RC. Sa puissance d'expression est grande, notamment parce qu'elle permet de représenter des informations incomplètes[3]. C'est une propriété fondamentale puisqu'une base peut difficilement encoder exhaustivement les connaissances d'un domaine.

---

[3] L'usage de la quantification existentielle et de la disjonction permet d'exprimer des connaissances incomplètes au sens où l'on peut ne pas préciser des "détails" soit par souci de généralité et de confort, soit parce qu'on les ignore.

Mais si l'on utilise pleinement le pouvoir d'expression de la logique des prédicats, les algorithmes nécessaires à sa mise en oeuvre informatique se révèlent très vite trop complexes, même pour accomplir des déductions simples. Une grande partie des approches en RC va donc se ramener à couvrir des sous-ensembles de cette logique garantissant de bonnes propriétés de calcul, au détriment, éventuellement, de la complétude. Un des grands enjeux de la recherche en RC est la recherche de cet équilibre entre puissance d'expression et manipulation (*tractability*) informatique [LEVE 87][4].

6.4.2.3. Réseaux sémantiques, Frames

Les différentes variantes de formalismes basés sur les réseaux sémantiques s'accordent généralement sur le fait que le domaine de connaissances doit être représenté sous forme de graphe étiqueté dans lequel les noeuds représentent les **concepts** du domaine et les arcs représentent les relations/liens entre ces concepts. Les relations sont appelées **rôles** (ou attributs).

Les noeuds représentent des individus du domaine ou des classes d'individus. Ils se traduisent habituellement en logique sous forme de prédicats unaires (cf infra la notion de types). Les liens se traduisent sous forme de prédicats binaires. Un lien spécial est rencontré dans tous les réseaux, celui rattachant une sous-classe à sa super-classe (ou un sous-concept à son super-concept), le lien IS-A (EST-UN). Il correspond aux relations hiérarchiques ou de subsomptions entre concepts.

Un concept A **subsume** un concept B s'il est "plus général", si donc B est un sous-concept de A, ou A lui-même. Si les concepts dénotent des ensembles d'individus, alors B est inclus dans A[5].

Des algorithmes efficaces de parcours des liens EST-UN calculent les relations de subsomption et gèrent l'héritage de propriétés des super-concepts aux sous-concepts. L'héritage évite donc la duplication des informations.

---

[4] Les termes "grande puissance d'expression de la logique des prédicats" ne doivent pas être compris de façon absolue. Beaucoup de connaissances se représentent difficilement en "simple" logique du premier ordre. Que dire alors des possibilités de manipuler informatiquement ces représentations !

[5] On remarquera la confusion des notations à propos de la subsomption. A subsume B a été symbolisé, dans les grammaires d'unification, A $\subseteq$ B, alors que la relation de subsomption dans le cas évoqués précédemment traduit la relation d'inclusion B $\subseteq$ A.

On a beaucoup discuté de la sémantique du lien IS-A. Sous un même terme se trouvent souvent confondus des relations très différentes. La distinction la plus classique, qu'il est necessaire de faire, est celle entre le lien reliant un individu/instance à sa classe (relation d'appartenance) et le lien reliant une classe à sa super-classe (relation d'inclusion) (pour les autres, voir [BRAC 83]). Dans le premier cas, le premier concept représente un individu (on parle souvent de **concept individuel**); dans le second, les deux concepts en relation représentent des classes d'individus.

Un des problèmes habituels rencontrés lors de la description d'un domaine à l'aide d'un réseau, est celui de l'abscence de critères de décision précis permettant de savoir si une information doit être représentée par un concept ou par un rôle. Le cas typique en langage naturel est celui de la traduction des verbes (cf annexe pour illustration). Les chercheurs en IA se sont, en effet, surtout préoccupé de développer des formalismes plutôt que des outils méthodologiques d'utilisation de ces formalismes ([SCHM 86] aborde ce point à propos de BACK).

Les avantages reconnus aux réseaux sémantiques sont leurs techniques efficaces de parcours de graphe mises en oeuvre pour les calculs des relations d'héritage et leur bonne adéquation expressive grâce à cette représentation graphique (cette dernière qualité est à nuancer compte tenu des remarques du paragraphe précédent).

Les inconvénients notoires sont les manipulations directes des structures par l'utilisateur que beaucoup de systèmes autorisent (l'utilisateur peut ainsi se redéfinir ses propres algorithmes de parcours de graphes). Le comportement du système devient alors souvent imprévisible. D'autre part les réseaux ne correspondent, dans leur version standard, qu'à un sous-ensemble très restreint de la logique des prédicats. Ce pouvoir d'expression trop limité se prête mal à la représentation d'informations incomplètes.

Les frames (prototypes) constituent une évolution par rapport aux réseaux grâce aux possibilités offertes de description des rôles et aux attachements procéduraux.

Les **descripteurs** sont des informations attachées à un rôle:

- méta-informations sur sa valeur: valeur par défaut, type, mode d'héritage.
- contraintes sur sa valeur: cardinalité, restriction sur l'ensemble des valeurs possibles.
- procédures attachées au rôle et déclenchées en cas de modification de valeur par exemple.

Descripteurs et attachements procéduraux donnent aux frames une bonne puissance d'expression et leur confèrent une bonne adéquation expressive pour représenter les situations prototypiques (scripts) en utilisant notamment les valeurs par défaut.

Si le mélange d'informations déclaratives et procédurales est attrayant, il a, par contre, l'inconvénient de ne pas donner aux frames une sémantique claire. De plus la notion de valeur par défaut est difficile à maîtriser. Des exemples célèbres [BRAC 85d] ont montré comment un système pouvait considérer tout prototype A comme spécialisation d'un prototype B (donc tel que A EST-UN B). Il suffit pour cela d'écraser au niveau de A la valeur des attributs caractérisants B et ajouter ceux qui caractérisent A[6].

Une autre source de confusion est la possibilité (et souvent la nécessité) de classer les concepts "à la main" parce que le système ne peut calculer automatiquement certaines relations de subsomption. Par exemple, le prototype "information ayant pour origine un et au plus deux processeurs (*info générale*)" subsume "information ayant pour origine deux processeurs (*info redondée*)". Si ce type de description complexe n'est pas possible dans le langage de description, l'utilisateur doit créer un prototype artificiel *info redondée* muni d'un descripteur *processeur* valué à deux (exactement), et le placer lui-même dans le réseau. Mais comment dire alors au système que cette description caractérise le prototype et ne peut être écrasée ?

6.4.2.4. Formalismes Hybrides

Chaque formalisme de RC a, en quelque sorte, sa niche. Il a été spécialement conçu pour représenter un type donnée de connaissances. Les formalismes hybrides essayent de faire coopérer plusieurs systèmes de représentation afin d'augmenter la puissance d'expression globale du système et pallier les défauts de chacun. Des formalismes intègrent ainsi règles de production et frames.

Les langages hybrides, qui nous intéressent ici, sont ceux du type de BACK, KRYPTON [BRAC 85c] et KL-TWO [VILA 85]. Dans ces systèmes le terme "hybride" réfère surtout à la séparation nette entre deux types de connaissances, terminologiques et assertionnelles. Un point important en RC est la détermination des statuts des connaissances représentées : statut épistémique (s'agit-il de croyances, hypothèses, définitions,..?), statut assertionnelle (ces connaissances sont-elles toujours vraies, connaissent-elles des exceptions, vraies avec un facteur de certitude ?). Ces langages hybrides représentent, dans leur partie terminologique (souvent appelée TBox), les définitions intensionnelles de concepts. Ainsi le concept *info redondée* sera défini dans la TBox et classé par rapport au concept *info générale* en tenant uniquement compte des descriptions respectives de chacun, sans qu'il soit question de l'existence ou non de telles infos. Le statut assertionnel des concepts est défini uniquement dans la partie assertionnelle du système, la ABox.

---

[6] Ce point recoupe la vieille question de la détermination des attributs caractéritiques d'un prototype, attributs qui devraient alors être protégés.

La partie **TBox** de ces formalismes s'inspire de KL-ONE. Un travail de clarification sur les frames a été entrepris, certains descripteurs ont été conservés, d'autres comme les valeurs par défaut écartés[7], ainsi que les attachements procéduraux. Un ensemble de primitives a été défini à partir duquel peut être construit les descriptions complexes des concepts et des rôles. Ces primitives se situent à un niveau intermédiaire entre un niveau logique (toutes se définissent et ont une sémantique logique) et un niveau conceptuel (elles sont de plus bas niveau que des primitives conceptuelles "à la Schank") de représentation des connaissances.

Le concept *info générale* peut se définir, par exemple, de la façon suivante (les primitives figurent en lettres grasses):

```
(44)
chose          = rootconcept1.
processeur     = primconcept1(specializes(chose)).
info_generale  = primconcept2(specializes(chose),
                   restriction(a_pour_origine(processeur,[1,2])).
```

Le réseau a un concept racine *rootconcept* que l'on peut renommé *chose*. *processeur* est un concept primitif, subsumé par (spécialisant) *chose*. La description d'un concept défini représente ses conditions nécessaires et suffisantes de définition, alors qu'un concept primitif n'est que partiellement déterminé par sa description qui ne représente donc que des conditions nécessaires. *info_generale* spécialise *chose* et est en relation avec *processeur* par le rôle de nom *a_pour_origine*, dont la cardinalité minimum est 1 et maximum 2.

A ce niveau de description l'héritage peut recevoir une définition rigoureuse. Il est alors possible de construire un **classificateur**, chargé d'insérer à la place optimum dans le réseau tout nouveau concept sur la base de sa description ou, à défaut, de rejeter cette description parce que contradictoire avec la base. En prenant systématiquement en charge le classement des concepts, le système apporte à l'utilisateur un moyen de vérifier la cohérence des connaissances qu'il a définies[8].

Si maintenant nous définissons *info redondée* comme un concept défini de la façon suivante:

```
(45)
info_redondee = defconcept(specializes(chose),
                        restriction(a_pour_origine(processeur,[2,2])).
```

le classificateur calculera automatiquement les liens de subsomption entre *info_redondee* et *info_generale* et classera le second comme sous-concept spécialisant le premier.

---

[7] D'autres niveaux de représentation ont été proposés par certains systèmes pour représenter les notions fondamentales de valeur par défaut et d'exceptions (cf [KAYS 88], [CAST 88] sur la gestion des exceptions dans les réseaux basée sur la logique des défauts que nous évoquons dans le dernier chapitre ; [PUJI 88] situe bien ce problème dans le cadre plus général des raisonnements non-monotones).

[8] A comparer avec les classifications hiérarchiques entre types que l'on doit faire "à la main" dans UCG et LIFE.

Le langage de la **ABox** est un sous ensemble de la logique des prédicats, permettant notamment de représenter certaines informations incomplètes. La ABox est en correspondance directe avec la TBox, puisque toutes les entités nommées dans la ABox doivent correspondre à des entités de la TBox.

```
(46)
uc1 - processeur.
uc2 - processeur.
uc3 - info_generale(a_pour_origine =
          and(card(1,1),or(uc1,uc2))).
```

uc3 est une info générale qui a pour origine un seul processeur qui est soit uc1, soit uc2.

Ces langages hybrides ont donc une sémantique claire. Et ici, la sémantique n'est pas une fin en soi. Elle a un rapport direct avec le comportement informatique du système. Le formalisme a une syntaxe proprement définie, une sémantique expliquant "le sens" des expressions bien formées du langage et le système se comporte de façon correspondante à ce qui est énoncé.

Qu'en est-il des problèmes entre puissance d'expression et manipulation informatique ? Les recherches sur les primitives de description ont montré que le problème du calcul automatique de la subsomption est un problème NP-complet, à moins que les primitives retenues ne soient triviales. Le débat sur la perte de généralité qu'il convient d'admettre pour avoir des temps de réponse acceptables est donc ouvert.

Les problèmes de complexité et manipulation informatique sont encore bien mal maîtrisés. Ainsi Krypton, de l'avis même de ses auteurs, est trop complexe et lourd d'utilisation. Cela s'explique en partie parce que la gestion de la ABox est confiée à un démonstrateur général de théorèmes et que la syntaxe des expressions admises est beaucoup trop riche. En RC manipulation informatique rime souvent avec algorithmes spécialisés sur des problèmes restreints plutôt qu'avec démonstrateurs logiques généraux.

Les auteurs de BACK n'autorisent qu'un sous-ensemble restreint de la logique dans la ABox: disjonction possibles mais pas n'importe où, limitations des formes de quantification,.... Sur ce sous-ensemble ils garantissent des temps de réponses acceptables "dans la majorité des cas".

Un des points qui différencie BACK de KL-TWO est celui de **l'expressivité équilibrée** entre la ABox et la TBox présente dans BACK, mais pas (ou peu) dans KL-TWO. Un tel équilibre correspond à une sémantique et une théorie représentationnelle réellement commune entre les deux parties du système. Dans KL-TWO, par exemple, les restrictions numériques (cardinalité des rôles) sont exprimables dans la ABox, mais ne correspondent à rien dans la TBox. Des problèmes similaires se posent concernant la négation, l'opérateur de disjonction, les restriction de domaines pour les rôles.

Un exemple de bonne correspondance entre ABox et TBox dans BACK est la notion de "compréhension" (*realization*). Des assertions dans la ABox peuvent créer de nouveaux concepts ou réorganiser le classement de concepts déjà existants dans la TBox. Ainsi l'assertion précédente concernant l'individu uc3 provoquera la création d'un concept *info générale ayant pour origine un processeur exactement* dont uc3 sera un représentant. Ce concept existe d'ailleurs dans notre domaine sous le nom d'*information non-redondées*. L'utilisateur peut renommer ainsi le nouveau concept généré par le système.

6.4.3. Terminologie propre au réseau

Nous donnons quelques indications supplémentaires sur la terminologie que nous avons utilisée dans ACTES pour rendre compte de l'implantation du réseau à l'aide de BACK. Cette terminologie, comme la représentation graphique donnée ci-dessous, s'inspirent de KL-ONE.

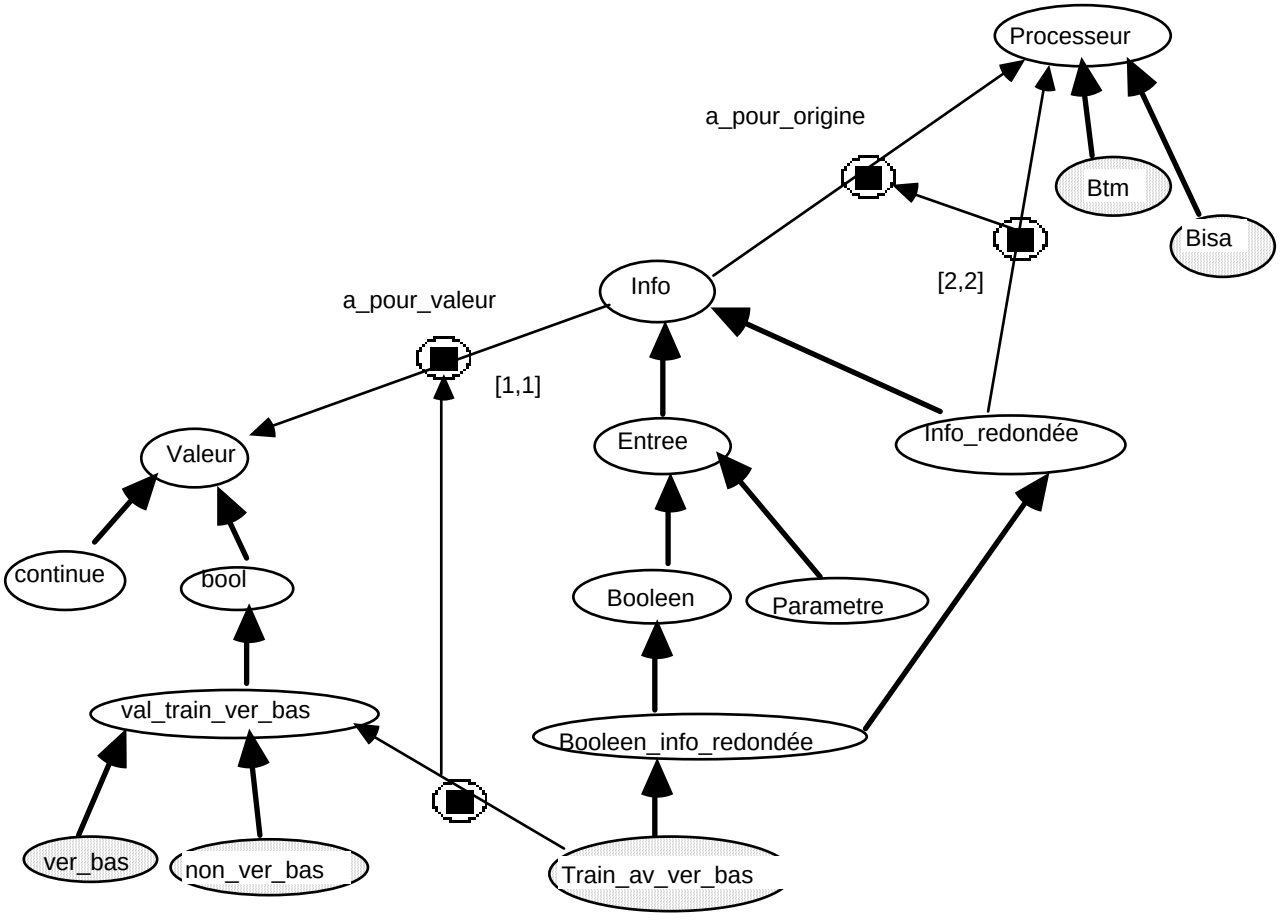


Fig 6.4: Description partielle et simplifiée du domaine

Dans ce réseau les noeuds du graphe sont les concepts, les arcs sont les rôles. Parmi les concepts, ceux figurant dans un ovale gris sont des concepts individuels: *Train_av_ver_bas*, par exemple. Les concepts individuels n'admettent pas de sous-concepts. On dira aussi qu'un concept individuel est une instance d'une classe. *Booléen-info-redondée* est la classe de *Train_av_ver_bas*.

Parmi les rôles, ceux marqués en traits pleins représentent les liens hiérarchiques entre concepts. Même si la représentation est la même, on distinguera éventuellement, le lien entre une instance et sa classe (souvent appelée lien SORTE-DE), du lien entre une classe et sa super-classe (lien EST-UN). Lorsque nous parlerons de sous-concept ou super-concept, on ne s'intéressera pas aux distinctions classes/instances. Le **sous-concept maximal** de deux concepts A et B est, parmi les concepts qui sont subsumés par A et B, celui qui subsume tous les autres.

6.4.4. Connexion ACTES-BACK

Il est possible, à partir de l'environnement ACTES, de se connecter à BACK[9] , de décrire son domaine et de récupérer (partiellement pour l'instant) cette description pour l'utiliser dans ACTES.

```
back_actes(Liste_rôles, Nomfichier)
```

Ce prédicat, disponible dans BACK, permet de sauvegarder les graphes correspondant à chacun des rôles de *Liste_rôles* dans un fichier. Ce fichier peut être chargé dans ACTES. Les liens de subsomption sont désignés par les liens SORTE-DE et EST-UN[10]

Pour plus de détails pratiques se référer au manuel d'utilisation [CHAN 89].

La correspondance entre les prédicats utilisés dans les DRS et les concepts du réseau est précisée dans la présentation de la grammaire. On trouvera un exemple dans le paragraphe suivant.

6.4.5. Types

6.4.5.1. Typer une variable

Les variables AVAG peuvent être typées. Le **type doit correspondre à un concept** dans le réseau BACK décrivant le domaine d'application. Les notations possibles sont:

---

[9] A condition de disposer de C_PROLOG.

[10] A l'heure actuelle, il n'est possible de récuperer que ces liens hiérarchiques. Cette information est suffisante pour le typage et l'unification étendue.

```
VAR:TYPE        ou        [VAR:TYPE,...]
```

Cela signifie qu'au cours de l'analyse VAR ne pourra s'unifier qu'avec un élément (atome/variable) dont le type est subsumé par TYPE.

Il est ainsi possible d'exprimer des contraintes sémantiques entre les mots. Si, par exemple, "positionner" ne s'applique qu'à des infos de sortie, et si on ne peut positionner une information qu'à une valeur, l'entrée lexicale du verbe peut se décrire:

```
(47) positionner:
...
<pred> = positionner(X1,X2)
<arg1> = X1:sortie
<arg2> = X2:valeur.
```

pred est le prédicat qui sera inséré dans la partie *condition* d'une DRS. *arg1* ne pourra s'instancier qu'à un élément dont le type est subsumé par *sortie* et *arg2* un élément dont le type est subsumé par *valeur*.

De façon à assurer une certaine cohérence entre la description du réseau et celle du lexique, le prédicat introduit par une entrée lexicale a autant d'argument que le concept correspondant dans le réseau a de rôles. De plus, le nom du prédicat est celui d'un concept du réseau et chaque type d'un argument du prédicat correspond au concept-portée d'un rôle. Ainsi *positionner* est un concept de la TBOX qui a deux rôles, le premier le reliant au concept *sortie* et le second au concept *valeur.*

Le type est également utilisé lors de la recherche de référents. Si un référent typé est introduit dans l'univers d'une DRS, le module de résolution vérifiera que le type de l'antécédent est subsumé par celui du référent.

6.4.5.2. Types implicites

Le système considère a priori que les atomes manipulés par l'opération d'unification étendue et les atomes présents dans les DRS sont typés. Il recherche leur type dans les informations du domaine de la façon suivante.

Un atome correspond à un objet du domaine représenté par un concept individuel du réseau. Son type est donc sa classe. Ainsi, l'information *Train_av_ver_bas* est de type *booléen-info-redondée*.

6.4.5.3. Unification étendue

Dans la plupart des grammaires d'unification, les traits terminaux ont des valeurs atomiques simples. La notion de typage relié à une description hiérarchique simple ou multiple est impossible. Cette notion est cependant indispensable pour décrire les objets d'un domaine. La description de ces objets impose des contraintes de sélection entre les mots du lexique et les catégories grammaticales de l'application. Ces contraintes doivent donc s'opérer au travers de l'opération d'unification.

Dans ACTES, nous disposons de la notion de typage reliée aux liens hiérarchiques de BACK. L'unification PROLOG ne prenant pas en compte les mécanismes d'héritage, nous avons défini une opération d'unification étendue gérant les contraintes hiérarchiques de types entre éléments (atome/variable) du langage AVAG. L'opérateur d'unification étendue est noté ':=:'. Son fonctionnement peut se décrire ainsi.

Nous noterons {X1 --> X} la substitution de l'élément X par l'élément X1.

Différents cas d'unification étendue entre deux éléments:

* X1 :=: Y1 Deux variables non typées s'unifient comme en PROLOG. Donc {Z --> X1} et {Z --> Y1}.
* X1 :=: Y1:T1 Une variable non typée et une variable de type T1 s'unifient et le résultat donne: {Z1:T1 --> X1} et {Z1:T1 --> Y1:T1}.
* X1:T1 :=: Y1:T2 Deux variables typées s'unifient s'il existe un sous-concept maximal T3 de T1 et T2 ; Le résultat donne : {Z1:T3 --> X1:T1} et {Z1:T3 --> Y1:T2}.
* a :=: X1 Un atome s'unifie avec une variable non typée ou un atome de la même façon qu'en PROLOG.
* a :=: X1:T1 Un atome s'unifie avec une variable typée si l'atome 'a' correspond à un concept individuel et que la classe T2 de 'a' est subsumée par T1. On a alors {a --> X1:T1}.

6.4.6. Exemple d'utilisation

Soit le lexique et la grammaire suivants:

```
(48)

utilise:
<cat> = verbe
<pred> = utilise(X1,X2)
<arg1> = X1:entree
<arg2> = X2:condition.

vitesse-sol:
<cat> = nominfo
<arg> = vitesse_sol.

gv -> verbe nominfo gp
...
<verbe arg1> :=: <nominfo arg>
```

sortie, *condition*, *vitesse_sol* sont des concepts. *vitesse_sol* est un concept individuel dont la classe est *vitesse*. *vitesse* est subsumée par *sntree*

L'analyse du groupe de mots "utilise vitesse_sol ..." unifie *X1* avec *vitesse_sol* après vérification des contraintes hiérarchiques.

7. ENVIRONNEMENT DU POSTE DE TRAVAIL

7.1. INTRODUCTION

L'organigramme ci-dessous présente les différentes étapes du traitement d'un texte (ou éventuellement de phrases isolées entrées au clavier) dans ACTES. Nous allons détailler la phase de découpage (module ACDEC), puis celle d'analyse (module HANNA) dans ses aspects d'aides après échec de l'analyse et stratégies d'analyse et enfin le module de résolution d'anaphores.

Notre présentation sera souvent critique. Critique des solutions originales que nous avons développées ou de celles que nous avons reprises à d'autres systèmes. Certains articles ou livres ont tendance à présenter des réponses partielles à certains problèmes comme des recettes miracles. Or leurs propositions s'appliquent mal à des problèmes de taille significative, sans parler des aspects qu'elles ignorent.

Ces solutions ont, par contre, l'avantage de la simplicité de mise en oeuvre. C'est la raison pour laquelle nous les avons introduites dans notre système. Car nous pensons que les points abordés ici: reconnaissance des constituants d'un texte (items ou structures d'ensemble), remédiations aux erreurs d'analyse, coopération de stratégies multiples pour la résolution d'anaphores, tous ces points devaient figurer, même tronqués, dans notre système. Chacun d'eux est un objet de recherche en soi. Cette présentation apporte les premiers éléments pour des travaux en ce sens.

Nous ne parlerons pas dans ce chapitre de l'environnement d'aide à la compilation du formalisme AVAG. Des informations le concernant se trouve dans [CHAN 89].

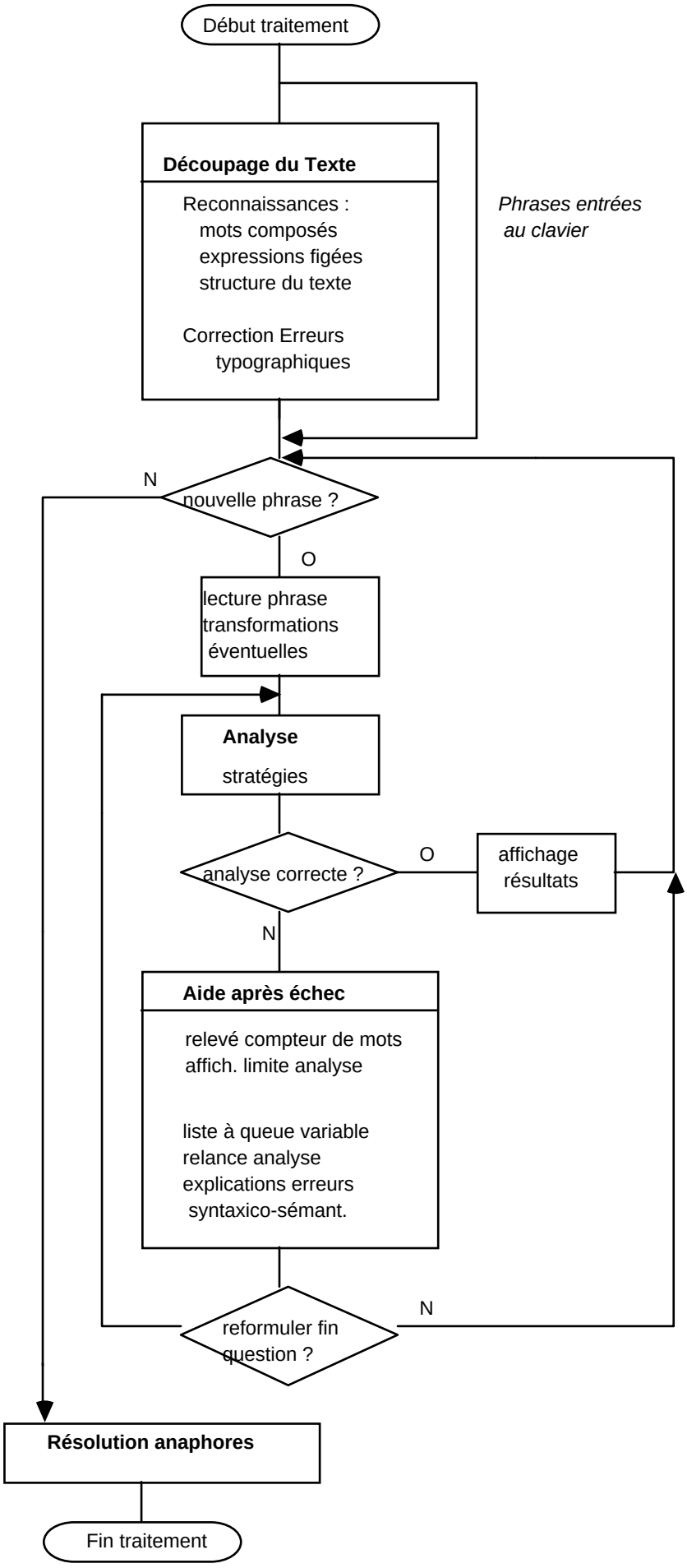

Début traitement
Découpage du Texte
Reconnaissances :
mots composés
expressions figées
structure du texte
Correction Erreurs
typographiques
Phrases entrées
au clavier
N
nouvelle phrase ?
O
lecture phrase
transformations
éventuelles
Analyse
stratégies
analyse correcte ?
O
affichage
résultats
N
Aide après échec
relevé compteur de mots
affich. limite analyse
liste à queue variable
relance analyse
explications erreurs
syntaxico-sémant.
reformuler fin
question ?
N
Résolution anaphores
Fin traitement

Fig 7.1: Organigramme général du traitement d'un texte

7.2. DÉCOUPAGE DU TEXTE

Le texte doit être découpé correctement préalablement à toute analyse. Ce traitement porte sur deux niveaux: reconnaissance de la structure du texte, reconnaissance des constituants élémentaires. Il est pris en charge par le module de découpage ACDEC (ACtes DECoupage).

7.2.1. Les constituants élémentaires

Les constituants élémentaires sont les mots simples ou composés, les signes de ponctuation.

Ces derniers peuvent apporter des informations ("-" est le séparateur des éléments d'une liste) mais peuvent aussi être ambigus (le ":" est annonciateur de liste ou introduit parfois le nom d'une information, la virgule encadre une apposition ou joue le rôle d'une conjonction de coordination, de séparateur d'éléments de liste).

Si l'utilisateur veut prendre en compte ces informations dans sa grammaire, il doit, au préalable, déclarer les signes de ponctuation au programme de découpage en attribuant à chacun d'eux un identificateur. Un signe sera alors considéré comme un mot simple et pourra être utilisé dans la grammaire à l'aide de son nom. A noter que tous les caractères non-alphanumériques (dont les signes de ponctuation) non déclarés sont ignorés lors de la lecture du texte.

Chaque nom composé ou expression figée est reconnue par le programme de découpage et transformé en un tout insécable. Les noms composés sont les noms d'informations ou de valeurs d'informations. ACDEC utilise les indications contenues dans le lexique généré à partir de AVAG pour les reconnaître en ajoutant des caractères soulignés, "_", entre chaque mot simple. Les valeurs des informations sont mises sous forme canonique: par exemple, "non verrouillée bas" sera transformée en "non_verrouillé_bas". On a en effet constaté que les accords de genre et de nombre sur les valeurs ne sont pas systématiquement respectés par les experts et qu'ils n'apportent pas d'information intéressante. Les expressions figées sont des groupes de mots comme "au lieu de", "c'est-à-dire", "au plus", etc... dont la décomposition en items individuels n'apporterait aucune précision intéressante en l'état actuel de l'analyse du texte. Chacune correspond à une entrée lexicale unique dans le lexique AVAG.

Précisons que ce traitement des mots composés et expressions figées est sans doute simplificateur, mais ne diminue pas le pouvoir ni la souplesse d'expression de l'utilisateur. ACDEC ne fait que reconnaître et figer comme un item ce que l'utilisateur a décrit par une unique entrée lexicale.

Le vrai problème se pose donc au moment de l'écriture du lexique et de la grammaire: quels groupes de mots doit-on figer, sur quels critères, pour faire quoi ? Au niveau du langage naturel non technique les régularités présidant à la formation de mots composés sont difficiles à mettre en évidence et les recherches actuelles qui s'intéressent au problème se contentent souvent de faire une description exhaustive des mots composés en vue de constituer des dictionnaires (cf les travaux de Gaston Gross et du LADL). Au niveau d'un sous-langage technique, des critères sémantiques propres au domaine permettent d'analyser une partie des mots composés (cf chapitre 5).

Mais élaborer un traitement automatique pour prendre en compte ces critères n'a de sens que si l'on a un objectif précis à réaliser. Ainsi, lors du développement du lexique et de la grammaire des textes de référence, nous avons décrit chaque nom d'information par une entrée lexicale. Nous l'assimilons à un nom propre. Chercher à analyser automatiquement le groupe de mots le composant n'aurait de sens que si l'on voulait, par exemple, augmenter automatiquement les connaissances du domaine [HIRS 88]: les nouveaux mots reconnus par la grammaire seraient introduits comme nouveaux concepts et placés correctement dans le réseau grâce au classificateur de BACK. Ce traitement pourrait être une extension à notre système.

D'autres expressions sont, par contre, analysées mot à mot. Ainsi dans les groupes nominaux comme "l'info d'entrée", "l'info de validité", dans les appositions comme "les infos trappes", "la commande train", le deuxième substantif est reconnu comme complétant le premier. C'est souvent le terme le plus informatif: on peut dire, en effet, "l'entrée", "la sortie", "les trappes" (cette dernière expression pouvant désigner le nom d'une information comme la pièce mécanique d'un avion).

7.2.2. Reconnaissances d'erreur d'inattention

Un premier traitement des erreurs est effectué dans ACDEC. L'intérêt est de ne transmettre à l'analyseur HANNA que des phrases dont les éléments, mots simples, composés, expressions figées, ont tous été reconnus comme référant à des entrées lexicales.

Les erreurs[1] rencontrées dans les textes sont généralement classées en trois catégories[2] :

- inattention ou mauvaise transcription

- méconnaissance de la langue

---

[1] Par erreur, on entend souvent tout phénomène entravant l'analyse correcte d'un texte par un système automatique. Nous n'inclurons pas ici, les problèmes dues à une mauvaise structuration du texte.

[2] La thèse de Fouqueré [FOUQ 88] présente une étude détaillée de ces types d'erreurs et un exemple de système d'analyse tolérante.

- modification intentionnelle de la langue

La première comprend les suppressions, ajouts, substitutions, inversions de caractères dans un mot. La suppression d'espaces ou de signes de ponctuation. D'après les statistiques, elles représentent une grande partie des erreurs dans les traitements de textes (au sens large) et sont essentiellement causées par l'utilisation du clavier.

C'est cette catégorie d'erreurs (moins les erreurs correspondant à des suppressions d'espaces, plus celles comprenant les fautes d'accentuation citées ci-dessous) que nous traitons dans ACDEC. Les mots sont considérés comme des chaînes de caractères et le traitement modifie un ou plusieurs caractères d'un mot jugée erroné de façon à atteindre une entrée lexicale. En cas de réussite, l'entrée est proposée à l'utilisateur. En cas d'echec, le programme se borne à signaler le mot comme inconnu. Ce traitement d'erreurs est tout à fait classique dans les traitements de textes du commerce. La procédure est relativement aisée à écrire en Prolog tant que le dictionnaire n'atteint pas une taille trop importante. Dans ce dernier cas, il faut structurer le dictionnaire et gérer des techniques d'accès appropriées, ce qui se fera de façon moins efficace en Prolog qu'avec des langages informatiques plus classiques.

La seconde catégorie inclut les erreurs phonétiques (substitution d'une syllabe par une autre correspondante au même phonème), les erreurs d'accentuation, les erreurs syntaxiques (fautes d'accord, par exemple), les erreurs sémantiques (constructions violant des contraintes sémantiques du domaine).

Les outils de remédiation à ces erreurs nécessitent l'utilisation d'une grammaire. Nous en reparlerons dans la partie suivante.

La troisième catégorie comprend notamment la création de mots ou expressions nouvelles. C'est un problème que nous n'abordons pas ici, mais qui deviendrait vite critique si notre système devait être étendu pour analyser un nombre important de textes. Traiter ce type d'"erreurs" est la seule façon de mettre à jour semi-automatiquement, en dialoguant avec l'utilisateur, un lexique (cf remarque ci-dessus sur les mots composés).

7.2.3. La structure du texte

Parallèlement à l'identification des constituants élémentaires du texte, ACDEC reconnaît sa structure : titres (du chapitre, des parties), parties, paragraphes.

La structure du texte apporte des métaconnaissances sur les spécifications : comment faut-il utiliser les informations (valeur, validité, dépendances entre informations,...) ? comment les règles sont-elles décrites (cas général, exceptions de niveaux différents) ? Ces métaconnaissances sont utilisées pour la génération des règles de défaut (cf chapitre 8).

De façon plus générale, connaître la structure d'un texte est indispensable pour pouvoir l'analyser. Un système de traitement automatique du langage naturel doit savoir de quoi on parle à tel endroit du texte. La structure de la représentation sémantique intermédiaire en dépend et cette représentation conditionne fortement, à son tour, les mécanismes mis en jeu dans la résolution d'anaphores (cf infra).

7.2.3.1. Comment s'opère la reconnaissance de la structure ?

ACDEC fournit des explications en ligne sur les paramètres utilisés pour déterminer les différentes structures (caractère de fin de phrase, de fin de partie, compositions d'un titre, noms des parties), et sur la composition générale du texte que le programme est en mesure d'analyser (grammaire BNF d'un texte). Les paramètres sont, bien sûr, modifiables par l'utilisateur. En revanche, la grammaire du texte ne l'est pas encore.

L'utilisateur compose son texte, en utilisant les paramètres précédents et les enchaînements entre sous-structures indiqués par la grammaire BNF.

7.2.3.2. Limites d'un tel mode opératoire

Signalons d'abord que nous avons été obligés de redéfinir un certains nombre de termes qui renvoient à des notions essentielles des systèmes de traitement de textes. Mais il n'est pas possible de laisser l'utilisateur écrire son texte avec de tels systèmes (beaucoup plus sophistiqués qu'un simple éditeur de texte ASCII) car les traitements de textes commercialisés ne sont pas ouverts et ne permettent pas de récupérer aisément les structures (ou plus exactement les composantes matérielles des structures textuelles) qu'ils manipulent : marques de titres, de paragraphes, de notes, d'indentation, de soulignement,...). Ces systèmes sont fermés car ils ne distinguent pas le plan conceptuel de cette description des textes du plan physique d'exécution[3].

Le mode opératoire de reconnaissance de la structure d'un texte est figé car la grammaire du texte est encodée de façon procédurale et implicite dans le programme ACDEC. Nous ne pouvons, aujourd'hui, que signaler à l'utilisateur ce que le système est en mesure de comprendre, sans lui donner les moyens d'écrire sa propre grammaire.

---

[3] Sur ce point, on peut lire l'article de Virbel [VIRB 82]. Pour mettre en évidence ce niveau conceptuel, il introduit la notion de fonction textuelle qui met en rapport les composants matériels des structures textuelles avec leur fonction d'utilisation (éditions de textes, par exemple). Ces fonctions textuelles peuvent s'élaborer en étudiant les actes de discours à portée méta-discursive (comment parle-t-on d'un texte, de telle ou telle partie, que signifie "introduire", "énumérer", "conclure", "noter", ...). Disposer de telles fonctions pourraient permettre d'assister l'utilisateur lors de la conception d'un texte en représentant, par exemple, ses métaconnaissances sous formes de règles

Il manque aussi un niveau de description dans les connaissances du domaine en rapport avec la structure du texte. L'utilisateur peut représenter certaines connaissances du domaine se rapportant au contenu des textes (cf chapitre 6). Nous ne savons pas comment lui permettre de représenter de façon déclarative les connaissances liées à la structure du texte[4], connaissances fondamentales pour la construction des règles (cf chapitre 8). Un exemple de texte "découpé" par ACDEC et des différents paramètres utilisés par le programme est donné dans [CHAN 89].

7.3. L'ANALYSEUR HANNA

7.3.1. Aides après échec de l'analyse

Deux types d'aides sont apportés à l'utilisateur en cas d'échec de l'analyse : visualisation des contraintes sur les traits qui n'ont pas été respectées, proposition du mot suivant.

7.3.1.1. Proposition du mot suivant

En cas d'échec dans l'analyse le système indique à l'utilisateur les N premiers mots de sa phrase qui ont été analysés correctement et lui propose de choisir un N+1 ième mot dans un ensemble généré par synthèse à partir de la grammaire. Cette tactique s'inspire de celle utilisée par Colmerauer dans l'interface ORBIS [COLM 82].

Expliquons en le principe à partir d'un exemple. Si l'utilisateur propose la phrase:

```
(1) * On utilise une sortie.
```

la grammaire reconnaît les trois premiers mots "on utilise une"' mais ne peut accepter "sortie" car dans notre domaine le verbe "utiliser" ne s'applique qu'aux infos de types *entrée* ou *paramètre*.

La phrase (1) est lue sur l'entrée courante, le clavier ici, transformée en liste de mots indexés. C'est cette liste qui est proposée à la grammaire :

```
(2) [ (on,1), (utilise, 2), (une,3) , (sortie, 4), (pOINT, 5)]
```

Après détection de l'erreur, le système constitue une nouvelle liste à queue variable avec les trois premiers éléments corrects de (2) et relance l'analyse.

```
(3) [ (on,1), (utilise, 2), (une,3) | VAR]
```

Supposons que la grammaire objet soit la DCG simplifiée[5] de la figure 7.2:

---

[4] A l'exception peut-être (mais elle est limitée) des règles pragmatiques qui explicitent dans quelles parties rechercher les antécédents de certaines anaphores (cf infra).

[5] Cet exemple représente bien la façon dont est traité le problème de la proposition du mot suivant dans ORBIS et dans notre système, mais est simplifié pour les besoins de l'exposé.

```
  ph(L, X0, X1) :-
     freeze(T,elem(T, [entree, param])),
     pronom(L1, X0,X2),
     verbe(L2,X2,X3),
     art(L3, X3,X4),
     nom(L4, X4,X1).
                        /* L, L1, ...,L4, sont les structures de traits
non détaillées. Il y a accord de type entre L2 et L4, et T est l'argument
de L2 correspondant au type  La primitive freeze diffère la vérification
de l'appartenance de T à la liste [entree, param] jusqu'à ce que T soit
instancié */

  nom(L,X1,X2):- mot(nom, L, X1,X2).
  art(...
  verbe(...

  mot(CAT,L, xxx) :- !, lex(MOT, CAT, L), write(MOT), fail.
  mot(CAT,L, [MOT|R], R) :- lex(MOT, CAT, L).


  lex(sortie, nom, [fem, sing, sortie]).
  lex(entree, nom, [fem, sing, entree]).
  lex(parametre, nom, [mas, sing, param]).
```

Fig 7.2: DCG, simplifiée afin d'expliquer le principe de proposition du mot suivant

Dans la règle *ph*, après l'analyse des trois premiers mots, la liste d'entrée transmise au littéral nom dans l'argument X4 est variable : c'est la queue de liste de (3). *nom* appelle le littéral *mot*. La première clause du paquet *mot* s'exécute puisque le troisième argument de *mot*, variable à l'appel, s'unifie avec *xxx*. Le prédicat *lex* permet d'accéder au lexique en connaissant la catégorie. Il ne réussit que pour les mots ayant des structures de traits qui satisfont les contraintes posées dans *ph*, contraintes d'unification entre les Li et contraintes d'appartenance à la liste *[entree, param]* pour le trait *type*[6]. Si un mot satisfait ces contraintes, *lex* réussit, donc l'évaluation se poursuit dans la première clause du paquet *mot*, le mot est écrit, puis un échec forcé (*fail*) permet d'afficher tous les mots satisfaisant les conditions ci-dessus.

Après l'analyse de "on utilise une", le mot "entrée" est affiché, ... mais pas le mot "paramètre", car son type est correct, mais pas son genre. Ainsi l'utilisateur n'aura pas toutes les solutions intéressantes. Cette façon de procéder oblige l'utilisateur à fixer le genre de l'article qui n'est pas (ou peu) pertinent. Il aurait sans doute voulu demander: "quelles sont les constructions possibles commençant par *on utilise un/une* ...? ".

La solution du problème réside dans la distinction entre contraintes syntaxiques et contraintes sémantiques et dans la possibilité de relâcher les premières lors de la nouvelle analyse.

Outre ce problème, le traitement proposé par Colmerauer a d'autres inconvénients :

- La méthode ne permet de synthétiser que le mot suivant. Elle ne peut donc rechercher des mots composés qui ne seraient pas représentés sous forme d'une unique entrée lexicale.

[6] Rappellons que ne figurent dans les structures de traits que les valeurs des traits, pas leur nom.

- Si la catégorie, à partir de laquelle s'effectue la synthèse, comporte beaucoup d'entrée lexicales, l'utilisateur verra apparaître une profusion d'informations sur son écran sans possibilité de discrimination.

- faire deux analyses pour, seulement, synthétiser le mot suivant est couteux et redondant. Ainsi, dans notre exemple, le système analyse inutilement deux fois "on utilise une".

- Même si l'utilisateur reprend un des mots proposés par le système pour continuer à formuler sa phrase, il se peut qu'au bout du compte ce choix aboutisse à une impasse car le système n'a qu'un regard en avant limité.

Les deux premiers points peuvent peut-être être résolus par des procédés appropriés, le troisième, non. Le problème de fond de ce traitement est qu'il ne fait que proposer un cheminement pas à pas incertain, alors que l'utilisateur aimerait avoir une vision globale de la couverture (surtout sémantique) de la grammaire, sans détail superflu.

C'est ce que proposent certaines interfaces en langage naturel, dites "guidées par menus", pour les bases de données [TENA 83]. L'utilisateur dispose sur son écran d'un ensemble de menus, correspondant aux principales catégories à partir desquelles des phrases compréhensibles par le système peuvent être formulées. Avec cette vision globale des choses, il peut plus aisément s'exprimer.

7.3.1.2. Dépistage d'erreurs dans l'analyse

Pour faciliter le diagnostic d'erreurs (au niveau du mot) dans l'analyse, HANNA repère le mot à partir duquel l'analyse a échoué, affiche les valeurs des traits de ce mot données par le lexique et celles attendues au niveau de la grammaire avant que le mot ne soit analysé. C'est la visualisation de la différence entre valeurs attendues et valeurs effectives pour les structures de traits d'un mot qui peut aider au dépistage d'erreurs.

Reprenons l'exemple de la figure 7.2 avec la phrase erronée "on utilise un sortie". Modifions la première clause du paquet *mot*.

```
(4)
mot(CAT,Traits, xxx):- write('traits attendus: '),
     write(Traits),
     recherche_traits_mot_err,
     !,
     lex(MOT,CAT,L), write(MOT), fail.
```

A l'appel de ce prédicat, une partie de la structures de traits *Traits* est déterminée. L'analyse de "on utilise une" a fixé les valeurs du nombre et du genre : *Traits* est alors égal à *[mas, sing,_]*. *recherche_traits_mot_err* affiche la valeur des traits du mot "sortie", à savoir: *[fem, sing, sortie]*. L'identification de l'erreur se fait alors sans peine.

Ce procédé simple (et très simplifié ici pour les besoins de l'exposé) ne permet pas de diagnostiquer les violations de contraintes de valeurs de traits, lorsque les contraintes sont exprimées, non par unification directe dans les structures de traits, mais dans des prédicats Prolog différents de ceux représentant les catégories grammaticales. Ainsi l'erreur de type dans "on utilise une sortie" ne peut être visualisée puisque cette contrainte est gérée par un *freeze*. La nature même des mécanismes de co-routinage comme le *freeze* ou le *dif* ne permet pas de décider explicitement du moment où il faut déclencher les contraintes qu'ils gèrent[7].

7.3.2. Stratégies d'analyse

La stratégie d'analyse utilisée aujourd'hui dans notre système est celle de Prolog. Nous discutons de ses limites, des remédiations possibles et des liens entre stratégies d'analyse et grammaire (ou formalisme).

7.3.2.1. Les limites de la stratégie Prolog

La grammaire objet généré par le compilateur AVAG est de type DCG. Elle est donc directement exécutable en Prolog. La stratégie standard de Prolog est descendante, gauche-droite, avec retour-arrière (backtrack) chronologique. Cette caractérisation correspond aux trois choix opérés dans la façon de parcourir l'arbre de recherche construit lors de la résolution d'un ensemble de buts :

- choix sur les clauses d'un même paquet : dans l'ordre où elles sont écrites.
- choix sur les littéraux composant un sous-but : de gauche à droite.
- choix sur les noeuds de l'arbre en cas d'échec dans la résolution ou après obtention d'une solution : retour au dernier point de choix laissé en suspens.

Les raisons qui ont guidé les choix des implanteurs de Prolog comprennent l'efficacité et la simplicité des programmes constituant le moteur du langage et la minimisation de l'espace en mémoire centrale lors de la mise en oeuvre de ce moteur. Ces choix font de Prolog un puissant démonstrateur de théorèmes[8].

---

[7] Décider d'utiliser ou non ces mécanismes de co-routinage n'est pas simple. D'un côté leur utilisation permet de programmer de façon plus déclarative (on pose des contraintes dans un programme un peu comme on énonce un problème sans se soucier du moment où elles seront exécutées) et plus efficace (réduction de l'espace de recherche dans les problèmes à forte combinatoire). De l'autre, on perd une partie du contrôle lors de l'exécution et un nombre d'appels trop grand à ces primitives finit par diminuer l'efficacité du programme car elle oblige le système à gérer de nombreuses piles auxiliaires.

[8] aux limitations près, dûes à la syntaxe des clauses (les clauses de Horn ne permettent pas d'exprimer certaines formules logiques) et à la non complétude (construction d'arbres de recherches infinis en cas de récursivité gauche dans des clauses, par exemple).

Mais cela n'implique pas pour autant que cette stratégie soit la mieux adaptée possible à l'analyse du LN. Le fait que Prolog se prête bien à l'écriture de petites grammaires (syntaxe proche des grammaires hors-contexte, disponibilité d'un moteur d'inférences), n'implique pas qu'il ait le comportement souhaité sur des grammaires de taille plus conséquente, ni que cette stratégie soit adaptée à la résolution de certains problèmes d'analyses en TLN[9].

Quels sont les inconvénients dûs à une utilisation directe du moteur Prolog ?

- le backtrack provoque une continuelle redécouverte des mêmes faits. Des sous-constituants de phrases peuvent être analysés plusieurs fois à la suite de backtracks provoqués par des échecs ou lors d'analyses multiples d'une phrase.
- l'exploration systématique des points de choix sur les têtes de clauses est souvent coûteuse et inutile. Supposons, par exemple, que la grammaire contienne une description des différents groupes verbaux du français sous forme d'une quarantaine de règles[10] de tête gv. La reconnaissance du verbe de la phrase suffit souvent à déterminer le bon groupe verbal correspondant (ou un sous-ensemble restreint). Prolog essayera pourtant systématiquement les quarante groupes verbaux (à moins de disposer d'une procédure de coupe-choix à effet local, *cut* local).
- Les règles récursives à gauche conduisent à des boucles infinies. Les exemples ne sont pourtant pas rares de règles de grammaire s'exprimant naturellement sous cette forme.
- Les possibilités très limitées de contrôles de l'exécution de la grammaire restreignent beaucoup les aides après échec de l'analyse, comme nous l'avons vu. Le contrôle est d'autant plus difficile, dans notre système que nous utilisons des méta-prédicats sur la grammaire DCG pour gérer l'effacement de catégories ou leur conjonction.

7.3.2.2. Remédiations possibles

Une première façon de répondre serait de dire qu'il suffit de programmer avec attention. Il est possible en effet de contourner en partie ces problèmes, mais les pseudo-catégories que l'on est alors obligé de créer, pour éviter la récursivité à gauche , les points de choix trop nombreux, n'ont plus beaucoup de pertinence linguistique. Nous perdons le niveau de déclarativité que nous souhaitions avoir avec le formalisme AVAG et les problèmes de contrôle ne sont pas réglés pour autant.

---

[9]L'universalité de l'approche démonstration automatique pour résoudre un problème quelconque, n'implique pas que cette approche soit la meilleure, ni même tout simplement utilisable. Ainsi toute une classe de problèmes NP-complets à forte combinatoire sont insolubles en Prolog. Il est bien sûr possible d'écrire des programmes Prolog censés trouver des solutions, mais leur mise en oeuvre est impossible du fait notamment du parcours exhaustif, sans optimisation de l'arbre de recherche. C'est la raison pour laquelle les langages logiques avec contraintes [DINC 88] se développent aujourd'hui.

[10] C'était le nombre approximatif de règles dans la grammaire en chaîne du français traduite en AVAG [CHAN 87].

Il existe des réponses partielles à ces problèmes.

Pour éviter d'analyser plusieurs fois les mêmes constituants, il suffit de stocker chaque nouveau constituant dans une table (*chart*) en indiquant ses positions dans la chaîne d'entrée. Un méta-interpréteur simple, placé au-dessus de celui de Prolog, teste avant d'appeler un sous-but si le constituant correspondant n'a pas déjà été analysé en consultant cette table.

Pour limiter l'exploration inutile de points de choix laissés en suspens, nous avions dans une version antérieur de AVAG indexé chaque règle. Connaissant une sous-classe de verbe, il était possible d'accéder au groupe verbal, *gv(1,_,_)* par exemple, à l'aide de cet index. L'unification sur la tête de clause est beaucoup plus rapide (de 5 à 10 fois) que celle opérée en partie droite de règle. Toutefois, insérer proprement ce type de connaissances, de nature différente des traits syntaxico-sémantiques, dans le langage AVAG était problématique. Nous reviendrons sur ce point.

Pour traiter la récursivité à gauche, deux types de solutions sont envisageables. D'abord changer de stratégie d'analyse. Les stratégies montantes (*bottom-up*) ne génèrent pas de boucles infinies. Elles peuvent être utiles également en diagnostic d'erreurs car l'analyse étant guidée par les mots de la phrase, il est toujours possible, en cas d'échec dans la reconnaissance complète de la phrase, de visualiser les constituants partiels acceptables. Par contre, cette stratégie n'étant pas guidée par les buts a l'inconvénient, lorsque des mots ont plusieurs catégories, d'accéder à certaines entrées lexicales et de construire des analyses partielles qui s'avèreront finalement erronées. L'ajout de prédictions descendantes dans des stratégies montantes est bien sûr possible. Bâties sur ce principe les stratégies dites de *left corner* sont, aujourd'hui, couramment utilisées. Mais la limitations des possibilités de guidage descendant dans ces analyses montantes rend plus difficile la réduction des phénomènes de ré-analyse cités précédemment.

L'autre solution, baptisée *Earley Deduction*, vise à garder une stratégie générale montante, couplée avec des mécanismes de prédictions et l'utilisation d'une table pour limiter les ré-analyses. [PEIR 87] détaille cette stratégie adaptée des algorithmes de Earley développés dans les années 70 pour des grammaires hors-contexte.

Cette méthode évite (dans presque tous les cas) les phénomènes de bouclage et les analyses redondantes. Mais cela se fait au prix de la réquisition d'espaces mémoire importants pour le stockage des tables de constituants, comprenant, pour chacun d'eux, leurs positions et tout leur environnement (sinon les unifications seraient perdues). Prolog, lui, n'explorant qu'une possibilité à la fois ne sauvegarde que l'environnement courant et défait les unifications à chaque retour arrière.

7.3.2.3. Stratégies d'analyse et écriture de grammaires

Il n'y a donc pas de solution absolue en analyse. Un bon environnement de développement de grammaires devrait offrir à l'utilisateur le choix entre plusieurs stratégies.

Les différentes méthodes évoquées précédemment n'impliquent aucun changement dans la grammaire source. Seul le compilateur AVAG devrait suivant la stratégie retenue générer des variantes de code DCG. L'analyseur devrait bien entendu avoir plusieurs interpréteurs.

La construction de tels interpréteurs (en fait des méta-interpréteurs par rapport à celui de Prolog) est la seule façon d'espérer contrôler le déroulement de l'analyse. [ALLE 87] rappelle comment il est possible de faire des prédictions portant non pas sur le mot suivant, ni même plusieurs mots en avant, mais sur plusieurs constituants. Ces prédictions permettraient d'avoir des analyses presque déterministes et amélioreraient beaucoup la qualité de la synthèse par rapport à la technique du mot suivant.

Les méta-interpréteurs sont intéressants aussi pour une autre raison. Le développeur de grammaires a besoin d'exprimer des préférences pour les ambiguïtés lexicales, les rattachements de groupes prépositionnels, le choix de groupes verbaux,.... Ces connaissances peuvent se traduire par un simple arrangement des règles en tenant compte de la stratégie de l'analyseur, par l'utilisation de procédures Prolog disséminées dans le code DCG. Mais nous perdons alors la déclarativité du formalisme grammatical, son indifférence à l'ordre des règles. Le travail est de plus à refaire après chaque compilation.

L'autre possibilité est de reconnaître que les connaissances en jeu sont en partie des métaconnaissances, du genre:

```
SI <cat1 trait1> = a1 ALORS FAIRE_UNE_FOIS regle(i)
                                  OU regle(j)
```

Par souci de clarté, il vaut mieux ne pas les mélanger avec les conditions syntaxico-sémantiques des règles AVAG. Les contraintes exprimées ci-dessus ne sont d'ailleurs pas impératives comme les contraintes d'unification de AVAG mais traduisent seulement des préférences. Il serait donc plus pertinent de développer un langage d'expression de ces métaconnaissances destiné à l'interpréteur. L'utilisateur aurait donc deux palettes d'outils différenciés: l'une concernant ceux décrits dans AVAG, l'autre concernant le choix de stratégies d'analyse et des méta-règles de contrôle de l'exécution de la grammaire.

Pour le lecteur intéressé par les problèmes de stratégies d'analyse signalons, outre les deux références données précédemment, la thèse de Morin [MORI 85] et le livre de Winograd [WINO 83] qui consacre plusieurs chapitres au sujet. Les résultats présentés, dans toutes ces références, s'appuient souvent sur des algorithmes utilisés dans les compilateurs et interpréteurs chargés de reconnaître des langages artificiels à l'aide de grammaires hors-contexte [AHO 86].

7.4. MODULE DE RÉSOLUTION D'ANAPHORES

Il n'existe pas aujourd'hui de théorie unique et complète de résolution d'anaphores, même si l'on se restreint au seul problème des anaphores pronominales. Les théories partielles existantes s'échelonnent depuis celles syntaxiques décrivant une partie des références intraphrases, à celles intégrant en plus, sémantique et pragmatique afin de couvrir des références interphrases. De plus, depuis [CHAR 72], on sait que la somme des connaissances pragmatiques liées au domaine d'un récit(simple) nécessaires à la résolution d'anaphores complexes est importante. Ce qui tient , aujourd'hui encore, hors de notre portée le traitement des anaphores dans un système de compréhension de textes généraux.

Nous n'en sommes pas réduit pour autant à implanter des solutions ad hoc pour traiter des textes de domaines limités. Il est possible de concevoir une architecture prenant en compte les théories partielles de résolution d'anaphores. Cette idée d'architecture générale, indépendante du domaine et s'intéressant au contrôle de modules intégrant le minimum de connaissances d'un domaine spécifique, est défendue par plusieurs équipes développant des systèmes de traitement du langage écrit [RICH 88], [CARB 88], [WADA 87].

L'idée développée ici, reprise de Rich, est de concevoir le système de résolution sous forme d'un **tableau noir** ("Blackboard") [ENGE 88] associant différents modules reprenant chacun une théorie partielle gérant un sous-ensemble des phénomènes du discours. Nous étendons ce système par la prise en compte de l'***accessibilité logique***. La Théorie de la Représentation du Discours nous permet de traduire en partie la portée des quantificateurs du langage naturel et restreindre les antécédents accessibles à un référent. Nous avons déjà parler des limites de la DRT. Nous montrons ici comment en pallier certaines.

La première partie évoque quelques architectures types de résolution d'anaphores. La deuxième décrit celle du système ACTES en détaillant plus particulièrement la source de contraintes *accessibilité logique* et la DRT appliquée aux textes de notre domaine. La troisième partie distingue notre architecture d'autres approches modulaires/multi-stratégies. Enfin une trace de résolution d'anaphores est présentée en annexe.

7.4.1. Architectures de systèmes de résolution d'anaphores

Quelles architectures ont été développées dans des systèmes de résolution de référents ? Comment répondent-elles aux problèmes soulevés précédemment ? Voici quelques indications permettant de mieux situer notre approche.

7.4.1.1. Un algorithme basé sur des contraintes syntaxiques

Hobbs [HOBB 78] présente un algorithme de résolution des anaphores pronominales. Sa procédure de calcul inclut des contraintes syntaxiques bien connues et quelques contraintes sémantiques simples (ainsi dans la phrase "il court", "il" ne peut référer qu'à un objet ayant le trait d'animalité). Cet algorithme a été implanté dans nombre d'interfaces en langage naturel. Hobbs l'a testé sur des textes de natures très différentes et montré qu'il résout 90 % des anaphores pronominales. Par contre il ne peut être étendu pour traiter les 10 % de cas supplémentaires. Dans le même article, il décrit un second algorithme avec des contraintes sémantiques étendues prenant en compte le contexte des mots. Mais les problèmes de structuration du discours ne sont pas abordés.

7.4.1.2. Sélection par des filtres successifs

Guenthner [GUEN 83], partant de la constatation qu'il n'y avait pas à l'époque d'approche uniforme et large du traitement des anaphores, propose de sélectionner les antécédents d'un référent anaphorique par filtrages successifs : morphologique, syntaxique, sémantique, pragmatique. La pragmatique n'intervient qu'après échec dans la recherche d'antécédents. Il est donc difficile de traiter des textes de domaine ayant un fort contenu pragmatique. Son domaine d'application est celui des interactions avec une base de données. Cette approche est sans doute pertinente pour ce cas précis, mais peu applicable au traitement de textes. Les phénomènes de suivi du discours telle la gestion du focus ne sont pas abordés, les références définies et plurielles non plus. De plus, les exemples de la partie "Stratégies, connaissances multiples" du chapitre 3 tendent à montrer qu'il est plus intéressant de regrouper les connaissances (morphologiques, syntaxiques,...) par type de références.

7.4.1.3. Gestion du focus

Le système de compréhension de textes PUNDIT contient un module de résolution d'anaphores, développé par [DAHL 86] et basé sur l'idée de focus [SIDN 79]. Le focus est l'objet principal sur lequel porte le discours à un instant donné. Une bonne compréhension du texte dépend en partie de la faculté de détermination du focus. L'idée est de le rechercher dans une liste contenant les marqueurs du discours. Pour cela il faut mettre au point un bon algorithme permettant de classer par ordre de pertinence, les marqueurs introduits dans la liste. Dahl utilise cette liste pour chercher les antécédents d'un référent anaphorique: l'ordre de la liste est le critère de préférence servant à sélectionner un antécédent. Quelques critères d'accord syntaxico-sémantiques simples sont ensuite appliqués pour retenir définitivement un antécédent.

Dahl propose un algorithme de gestion du focus basé essentiellement sur des critères syntaxiques et montre qu'avec une telle approche on peut traiter de façon uniforme les anaphores pronominales, les élisions de groupes nominaux et les références implicites. Le problème, avec cette approche, est que la liste du focus est l'unique source proposant des antécédents pour un référent et que les critères de sélection dans cette liste sont trop restreints.

7.4.2. Architecture du module de résolution d'anaphores

Dans la présentation suivante nous reprendrons largement la terminologie employée par [RICH 88]. Notre architecture rassemble un ensemble de modules implantant chacun une théorie partielle spécifique. Ils communiquent en proposant des antécédents possibles pour un référent anaphorique et évaluent les candidats proposés par chacun des autres. Un **module gestionnaire** (MG) supervise l'ensemble du processus et gère au besoin les conflits. Le module gestionnaire est activé à la fin de l'analyse d'un texte[11]. Il résout les anaphores phrase par phrase.

La structure de représentation sémantique partagée par tous les modules est la DRS associée au texte en cours d'analyse. Cette DRS est structurée en sous-DRS représentant titre, parties, paragraphes, phrases et portée des quantificateurs. Les référents du discours sont accessibles dans la partie *univers* des DRS. Des informations supplémentaires concernant type, genre, nombre, etc... peuvent leur être associés.

```
[ M - T, G, N , C ,D, AC, R ]

M : référent
T type
G : genre
N : nombre
C : cardinalité
D : défini, indéfini
AC : anaphore/cataphore
R : rôle syntaxique (sujet, objet,..)
```

Fig 7.2 : Exemples de traits associées à certains référents du discours

La partie *condition* des DRS donne les contraintes sémantiques associées à ces référents.

---

11 Certains phénomènes de référence étant des cataphores, il est nécessaire d'avoir analysé tout le texte pour pouvoir les résoudre.

```
[X1 : sortie , fem,plu,2,
      def, ana, suj]
id1 : event
_____________________________

positionne(id1,X1,alarme)
```

Fig 7.3 : Drs associée à : "Les deux infos de sortie sont positionnées à alarme".

Outre la DRS du texte, les sources de connaissances accessibles à tous les modules sont les arbres syntaxiques des phrases analysées, la structure du texte (titre, parties, paragraphes) et les connaissances du domaine provenant du réseau BACK. Un schéma de l'architecture globale est donnée en figure 7.4. Chaque ellipse représente une *source de contraintes* qui pose ses propres contraintes sur le choix des antécédents d'un référent.

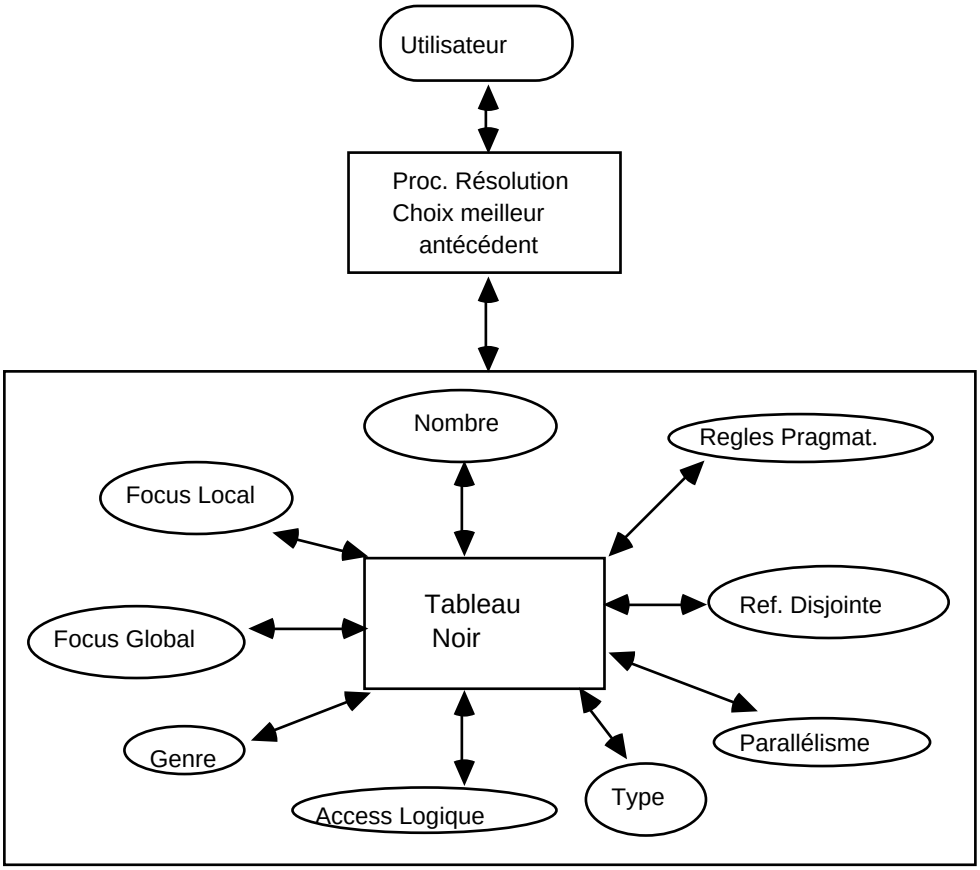


Fig 7.4 : Architecture générale du système de Résolution

7.4.2.1. Organisation générale des sources de contraintes

Chaque source de contraintes(SC) implante une théorie partielle de la résolution d'anaphores. Elle fonctionne comme une **boîte noire**. Le module de gestion ne connait d'une source que son type déterminant ses conditions d'évaluation (cf ci-dessous). L'interface de communication entre les modules est composé du référent courant et de la liste des candidats antécédents(CAND).

Chaque source peut fonctionner en **proposant** des antécédents pour un référent donné (comme les sources 'accessibilité logique', 'règles pragmatiques', 'focus global',...). Elle peut aussi fonctionner en filtre/**évaluateur** des antécédents proposés par une autre source. Certaines sources ne font qu'évaluer des contraintes sans jamais proposer de candidats ('genre', 'nombre', 'type',...).

Une source peut avoir accès à des connaissances qui lui sont propres : la source *Règles pragmatiques* gère ainsi ses propres règles.

Chaque source a un ou plusieurs **types d'évaluation** indiquant à quel moment du processus de résolution ou d'analyse elle doit être activée:

- type 1 : *pendant l'analyse syntaxico-sémantique du texte*. La source 'références disjointes' pose des contraintes d'exclusion entre des marqueurs du discours . Les règles décrivant les cas de référence disjointe sont stockées dans le module de résolution de référents. L'analyseur syntaxico-sémantique appelle ses règles pour poser ces contraintes, pendant le processus d'analyse. Par contre, ces contraintes de disjonction sont vérifiées pendant la résolution de référents (type 3).
- type 2 : *une fois par phrase*. Les règles pragmatiques, par exemple, ajoutent des connaissances en début de traitement d'un chapitre ou d'une partie. La rupture de partie ou paragraphe est gérée en début d'analyse de phrase.
- type 3 : *une fois par référent*. C'est le mode d'évaluation courant de presque toutes les sources de contraintes. Pour un référent anaphorique donné, chaque source propose des candidats ou évalue ses contraintes sur un antécédent proposé.
- type 4 : *si aucun candidat pour un référent n'a été retenu.* (cf source de contraintes *Ensemble*).

7.4.2.2. Procédure de résolution

La procédure de résolution de référents est déclenchée à la fin de l'analyse du texte. Les contraintes des sources de type 1 ont déjà été posées. Elles figurent dans la structure sémantique comme de simples littéraux. Le module de gestion a la main et commence la résolution à partir de la première phrase.

Pour chaque phrase:

* MG appelle chaque SC de type 2. Si un changement de partie, de paragraphe a eu lieu, une SC peut, par exemple, construire la DRS principale de la partie ou du paragraphe, ajouter des référents du discours dans son univers.

* Pour chaque référent:

- MG appelle chaque SC de type 3 avec, au départ, CAND = []. Toute SC qui ajoute de nouveaux référents du discours dans CAND les propose pour évaluation aux autres SC.

Si aucun référent n'est retenu, alors appel de la même SC de type 4, s'il existe, pour proposer de nouveaux antécédents.

- MG appelle la procédure choix-meilleur-antécédent. Si un antécédent est retenu, alors unifier le référent anaphorique avec l'antécédent et propager ces unifications dans toutes les conditions des DRS où ce référent apparaît.

La procédure choix-meilleur-antécédent fonctionne, pour l'instant, de façon très simple. Si CAND est vide, un message annonçant qu'aucun antécédent n'a été trouvé est édité. Si CAND ne contient qu'un antécédent alors antécédent et référent s'unifient. S'il y a plusieurs antécédents, l'utilisateur est invité à choisir lequel il retient (nous rediscutons le problème du choix d'un antécédent dans la partie suivante.

7.4.2.3. Problèmes posés par la pondération

Pour l'instant, chaque SC évalue ses contraintes sur une liste d'antécédents associée à un référent anaphorique en acceptant ou refusant chacun d'eux. La procédure peut sembler grossière, car certaines contraintes s'exprimeraient mieux en terme de **préférences** pour un antécédent donné au lieu d'un simple rejet/acceptation. L'exemple suivant tiré de [CARB 88] l'illustre bien :

La chef scout$_i$ associe Marie$_j$ à Susan, mais , la dernière fois, *elle$_i$ l$_j$*' avait associée à Nancy.

Carbonell énonce la règle de parallélisme syntaxique suivante : dans des phrases conjointes, adjacentes ou explicitement opposées, **préférer** l'antécédent qui préserve la structure de surface syntaxique de la première phrase.

Wada et Rich, pour résoudre ce problème, introduisent une pondération lors de l'évaluation. Lorsqu'une SC évalue ses contraintes sur un antécédent, elle lui affecte un certain coefficient de préférence. Une procédure calcule le score de chaque antécédent en fonction des coefficients attribués par chaque SC. Un seuil limite permet de rejeter un antécédent.

Cette tactique est certes attrayante, mais n'est pas sans poser des problèmes. Suivant quels critères établit-on ces coefficients de préférence : l'intuition, des tests sur un nombre significatif de cas de références ? Quelle est la bonne procédure de calcul du score lorsqu'une SC propose un antécédent suivant des critères contredisant ceux d'une autre SC ? Prenons un exemple pour illustrer ce dernier point.

Le texte prototype mentionne dans la partie 'CONTROLES-SUR-ALARME' "la condition sur la vitesse". La SC *accessibilité logique* propose les antécédents accessibles logiquement (cf partie suivante pour plus détails) et en particulier la condition portant sur une vitesse présentée dans la même phrase, à savoir "si la vitesse-sol est invalide également". La SC *règles pragmatiques* cherche des antécédents dans la partie précédente 'REGLE-PRINCIPALE'[12] et trouve une condition sur une vitesse : 'vitesse-conventionnelle inférieure à Vmini'. Notre système propose alors à l'utilisateur de choisir la bonne référence (ici la condition portant sur la vitesse-conventionnelle).

Ne pouvant accorder une priorité systématique d'une SC par rapport à une autre SC, nous avons décidé, en l'état actuel, de laisser l'utilisateur décider et n'avons pas encore implanté de procédure de pondération. Même si Rich reconnait des difficultés sur ce point et a amélioré sa procédure de calcul en introduisant la notion de coefficient de certitude associé au coefficient de préférence, les exemples présentés dans son article ne font pas intervenir de candidats proposés par plusieurs SC.

7.4.2.4. La source de contrainte 'accessibilité logique'

Le poste de travail du système ACTES a été développé dans un souci de généralité : d'où cette approche sémantique fondée sur la DRT. Mais les phénomènes de quantification sont peu nombreux dans les textes prototypes. Comment dans ce contexte appliquer la DRT ?

Dans l'approche de Kamp, le discours est essentiellement structuré par la traduction des quantificateurs du langage naturel. Ici, la structure la plus générale est imposée par l'organisation du texte en titre, parties, paragraphes.

La DRS générale du texte de la figure 1 est la **DRS du chapitre**. Son univers est l'univers principal du texte. Ses référents du discours (en partie ceux introduits par le titre) sont accessibles à tous les autres référents des DRS subordonnées. Ils représentent, en quelque sorte, le focus global du texte.

A **chaque partie** correspond une DRS. Dans son univers (univers principal de la partie) figurent les référents introduits dans les phrases de la partie par les noms d'informations, les déterminants définis, les *cas* d'une règle.

Le **nom d'une information** est son identificateur. Il est géré comme les noms propres dans les exemples de Kamp. L'accessibilité du marqueur correspondant est limité à la partie (et non à la DRS principale du texte). Chaque partie représente une description particulière du traitement, certaines informations n'ont qu'une portée locale relative à cette description.

---

[12] La partie 'CONTROLES-SUR-ALARME' décrit souvent des exceptions sur les conditions intervenant dans les prémisses de la règle principale. Une règle pragmatique indique donc de chercher des antécédents dans la partie précédente.

Les **articles définis** introduisent des références dont la portée dépasse souvent le cadre de la phrase (l'antécédent n'est généralement même pas dans la phrase précédente). D'où l'idée de placer les référents du discours qu'ils introduisent à un niveau plus général, celui de la partie.

La description de chaque règle du texte introduit systématiquement un **référent de type *cas*** représentant l'ensemble des conditions décrites dans les prémisses de la règle. La phrase après le titre, par exemple, décrit une règle comprenant 7 conditions. Le référent suivant sera placé dans l'univers principal de la première partie :

```
et(cond^1,cond^2,....,cond^7) -cas^1

cond^1 est un pointeur vers le référent de type condition représentant la
première prémisse de la règle et figurant dans l'univers de la DRS de la
partie 1.
```

On peut ainsi faire référence dans la suite du discours aux cas décrits dans la règle (cf "dans les autres cas").

Voici maintenant quelques indications sur la façon dont sont gérés l'accessibilité logique et les problèmes de continuation du discours :

- Toutes les conditionnelles du type "si ... alors" introduisent une implication entre deux DRS.
- Chaque nouvelle phrase d'une même partie vient augmenter (dans un premier temps) la DRS de la partie correspondante:
  * La figure ci-dessous représente la DRS partielle d'un chapitre après l'analyse de la première phrase de la deuxième partie. Si Dn+3 est la DRS correspondant à la phrase suivante, alors son univers augmente l'univers de Dpartie2 et ses conditions vont s'inscrire là où figure la croix 1.

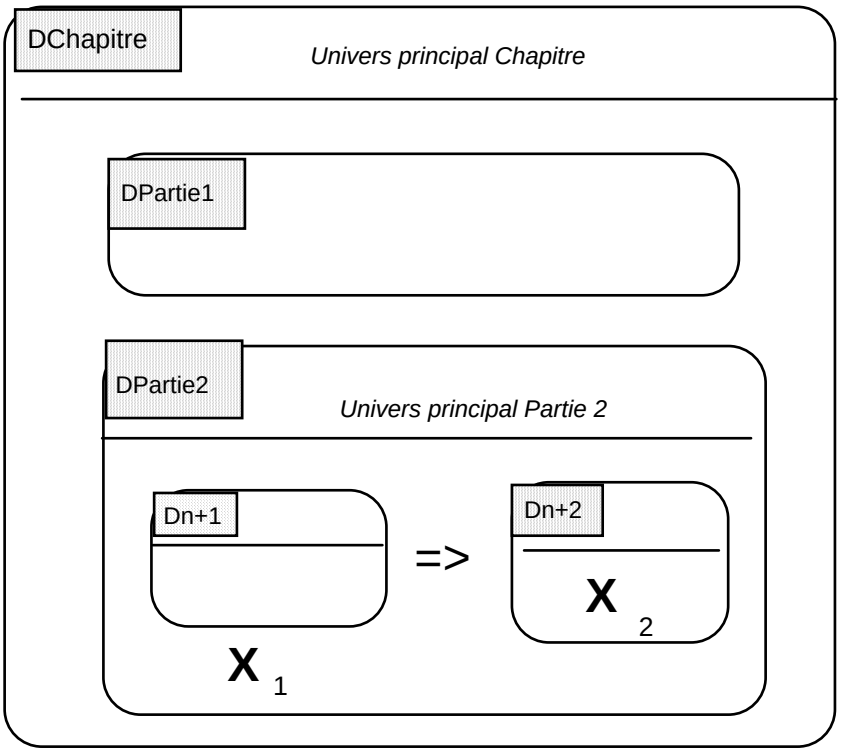


Fig 7.5: Deux continuations du discours possibles

* D'après cette figure, les référents accessibles à un référent de Dn+2 sont ceux de Dn+1, Dpartie2, et Dchapitre.

- Si, dans une nouvelle phrase, un référent co-réfère avec un référent de la phrase précédente non accessible logiquement (cf exemple donné dans les limitations de la DRT), alors on force l'accessibilité : la nouvelle DRS, Dn+3, augmente la dernière DRS introduite , Dn+2.

  En reprenant l'exemple précédent, l'univers de Dn+3 augmente cette fois, l'univers de Dn+2, et les conditions de Dn+3 viennent s'inscrire à l'emplacement désigné par la croix 2.

  Cette restructuration est aisée puisque l'ensemble des DRS est stocké sous forme d'arborescence. Il suffit alors de changer un pointeur pour déplacer un sous-arbre.

Les limitations présentes de la DRT nous ont amenés à traiter les contraintes sur la continuation du discours comme **une source de contraintes parmi d'autres** et donc à adopter cette architecture de tableau noir. D'autres sources de contraintes (en particulier des critères pragmatiques) peuvent proposer des antécédents considérés comme non accessibles dans le cadre de la DRT, ce qui entraîne une restructuration des DRS.

7.4.2.5. Autres exemples de sources de contraintes

Voici quelques indications sur des sources de contraintes différentes de l'*accessibilité logique*.

Genre, Nombre

Les sources *Genre* et *Nombre* utilisent les informations associées aux référents du discours pour sélectionner des antécédents. Elles ne proposent aucun antécédent. Elles opèrent donc en premier pour donner leur avis sur les candidats proposés par une source.

Type, Consistance

Les référents du discours (atomes ou variables), comme les atomes utilisés dans les parties *conditions* des DRS, sont typés. Leur type correspond à un concept du réseau sémantique dans lequel sont représentées les connaissances du domaine. Le type est soit explicitement mentionné, soit calculé à travers les liens hiérarchiques du réseau. Par exemple le référent *vit_sol* (cf trace de la recherche en annexe), correspondant à la vitesse-sol, est un concept individuel du réseau. Son type correspond à sa classe, le concept *vitesse*. Pour que deux marqueurs du discours puissent co-référer, il faut qu'ils aient des types compatibles (que l'un soit une super-classe de l'autre ou qu'ils soient dans une relation instance-classe).

La source *Consistance* opère des vérifications élémentaires : lorsque l'on cherche, par exemple "la condition sur la vitesse", le système va rechercher des conditions portant sur des vitesses en utilisant les prédicats contenus dans les parties *conditions* des DRS et les informations du réseau.

Ces deux sources n'opérant que des sélections sont appelés le plus tôt possible comme *Genre* et *Nombre*.

Règles pragmatiques

Les règles pragmatiques sont évaluées lorsque leurs préconditions sont satisfaites : elles peuvent ajouter des informations dans l'univers de certaines DRS qui seront reprises par la source *focus global* ou proposer des antécédents (cf l'exemple de "la condition sur la vitesse". En voici deux exemples :

```
reg_prag(type2,[apres(partie,titre)],
     [],
     [ajout_ref(univers,titre,chapitre)] ).
```

A la fin des recherches de référents du titre, augmenter la DRS principale du chapitre du contenu de l'univers du titre. Cette règle de type 2 ne sera évaluée qu'une fois par partie.

```
reg_prag(type3,[pendant(partie,cTRL_AL)],
      [type_ref(condition)],
      [rech_ref(partie,rEG_PAL)]).
```

Cette règle de type 3 est évaluée pour chacun des référents anaphoriques de la partie CONTROLES_SUR_ALARME : si le référent est de type *condition* alors rechercher antécédent parmi les référents de la partie REGLE_PRINCIPALE.

Focus local et Focus global

Les informations de type *sortie* mentionnées dans le titre du chapitre sont les objets principaux sur lesquels porte le traitement décrit dans la suite du texte. Dans certains textes, on fait référence au *traitement* bien que ce mot n'ait à aucun moment été introduit. Cette notion est implicite puisque chaque chapitre est la description d'un traitement. Toutes ces informations doivent être accessibles à partir de n'importe quelle DRS. Des règles pragmatiques les introduisent dans la DRS principale du texte et la source *focus global* les propose.

La source *focus local* n'agit ici qu'en cas d'échec dans la recherche d'un antécédent. Elle proposera alors le dernier antécédent auquel on a fait référence, dans la même phrase ou une phrase précédente. Si cet antécédent est accepté et n'est en principe pas accessible logiquement il pourra y avoir réorganisation dans la structure des DRS.

7.4.3. Comparaison avec d'autres approches modulaires/multi-stratégies

Le module de résolution d'anaphores de ACTES se situe dans la lignée d'autres approches modulaires et multi-stratégies: Carbonell, Rich, Wada. En voici quelques traits comparatifs et distinctifs.

Carbonell [CARB 88] a décrit la grande diversité (voire la finesse) des stratégies mises en oeuvre dans la résolution de référents. Son système applique d'abord les stratégies de contraintes de façon à sélectionner le plus efficacement possible les meilleurs antécédents, puis des stratégies de préférences pour opérer un choix. Le problème de savoir comment un discours introduit et structure les référents du discours qui seront des antécédents potentiels n'est pas abordé.

Notre approche est plus modulaire car une même source de contraintes peut opérer en proposant aux autres sources des antécédents parmi les référents du discours présents dans les DRS, et en sélectionnant ceux proposés par les autres sources. De plus, nous avons implanté des niveaux de stratégies différents (types d'évaluation) reliés à la structure du texte et aux problèmes de reprise après échec.

Notre architecture s'inspire de celle présentée par **[RICH 87]**. Mais, comme dans le cas précédent, nous identifions plus précisément ce que sont les objets du discours auxquels on peut référer et leur structuration. Ces informations sont fournies dans ACTES par les DRS et par la source de contraintes *accessibilité logique* (cette source n'avait été qu'évoquée par Rich).

D'autre part, dans les exemples présentés dans son article, une seule source propose des antécédents (même si, en principe toutes peuvent le faire). Le fait, dans Actes, d'avoir plusieurs sources qui agissent à ce niveau, nous a fait poser la question de la pertinence des critères de pondération devant opérer un choix parmi les meilleurs antécédents.

[WADA 87] utilise aussi les structures sémantiques DRS couplées à une grammaire d'unification ([Wada 86]). Les antécédents potentiels d'un référent sont strictement sélectionnés parmi ceux accessibles logiquement dans ces DRS. Pour opérer une sélection, il applique des filtres syntaxiques et lexicaux et détermine un ordre de pertinence (*salience ordering*) tiré des certains travaux sur le focus.

Dans ACTES, nous avons choisi de traiter l'accessibilité logique comme une source de propositions d'antécédents parmi d'autres et donc d'accepter des références ne respectant pas ces critères. C'est une façon de pallier les limitations présentes de la DRT afin de l'appliquer aux textes techniques du projet ACTES. Ce problème n'apparaît pas dans l'article de Wada : les exemples d'implantation restent dans le cadre habituel (pour la DRT) des 'donkey sentences'.

7.4.4. Conclusion sur le module de résolution

Le module de résolution d'anaphores du système ACTES permet de faire coopérer des sources de contraintes multiples ayant chacune leur propre stratégie dans un même système, au lieu de tenter de construire une méthode monolithique.

Parmi les limitations de notre système on peut relever l'implantation partielle (voire l'absence) de certaines sources de contraintes. Nous n'avons développé que le minimum nécessaire à la résolution des cas d'anaphores des textes prototypes de notre domaine.

En particulier les stratégies des sources *focus local* et *focus global* sont suffisantes pour traiter les textes qui nous intéressent, mais seraient trop naïves pour aborder la compréhension de récits. Un des idées intéressantes développées par [CART 87] est d'avoir montré dans son approche *shallow processing* de traitement des anaphores des textes, qu'il était possible et souhaitable de limiter au maximum l'accès aux connaissances du monde. Celles-ci sont bien sûr indispensables, mais restent très difficile à décrire et exploiter simplement. En contre-partie, il présente un développement fin des techniques de suivi du focus. [ALLE 87] a montré qu'une bonne gestion du focus global ([GROS 83]) dépend de l'implantation de stratégies de planification.

Quelle serait la place de cette planification dans notre architecture ? La remet-elle en cause ou peut-elle s'ajouter à un méta-niveau par rapport aux sources de contraintes, être placée dans le module de gestion par exemple ?

Malgré ces limitations, notre travail présente l'originalité d'avoir intégré dans une architecture modulaire une théorie du discours. Elle permet d'avoir une définition explicite des objets auxquels on peut faire référence, de traduire certains phénomènes de quantification souvent laissés de côté, d'avoir un environnement d'interprétation cohérent puisque les structures sémantiques générées par l'analyseur correspondent à la même théorie du discours. Nous avons pallié certaines insuffisances de cette théorie en la considérant comme un des modules de notre architecture. Le système a été expérimenté en utilisant plusieurs sources de contraintes proposant des antécédents, ce qui nous a permis de soulever les problèmes posés par la pondération des sources de contraintes.

8. EXTRACTION DE RÈGLES

"Nonmonotonicity appears to be the rule, rather than the exception, in much of what passes for human common sense reasoning"

Reiter R.: "A Logic for Default Reasoning" , Artificial Intelligence, 1980.

8.1. INTRODUCTION

Le système expert simulant les alarmes est à base de règles de production. Ces règles sont aujourd'hui écrites à la main après lecture des textes de spécifications. Plutôt que de générer directement de telles règles, nous proposons d'essayer de produire automatiquement des règles de défaut à partir de la représentation sémantique intermédiaire (les DRSs).

Ce choix est motivé par la correspondance entre la représentation du processus d'extraction des règles dans un texte et la non-monotonie qui se formalise bien dans la logique des défauts. De façon plus générale, le traitement automatique de textes en langage naturel s'assimile bien à un processus non-monotone[1]. Enfin, et surtout, nous disposons d'un cadre pour envisager la vérification interactive de cohérence locale des textes.

Dans une première partie nous rappelons brièvement ce qu'est la logique des défauts.

La deuxième partie donne une idée de l'intérêt de l'application de cette logique au domaine de ACTES et l'illustre par un exemple. Pour ce faire, nous décrivons de façon plus détaillée qu'au chapitre 5, les objectifs que pourraient avoir un expert en dialoguant avec notre système, nous formalisons la notion de traitement correspondant au chapitre d'un texte de spécification, puis indiquons sur l'exemple du texte de référence quels défauts pourraient être produits.

[1] Pour une application de la logique des défauts à d'autres problèmes en TLN, comme la quantification, voir [SAIN 88].

La troisième partie de ce chapitre indique comment étendre notre système avec les défauts. Trois étapes sont envisagées : production des défauts à partir de la représentation sémantique intermédiaire d'un texte, déduction sur les défauts, traduction des défauts en règles de production. La première et la troisème étapes sont discutées à partir de l'exemple présenté dans la deuxième partie. La traduction DRSs-défauts est détaillée plus particulièrement: algorithme de traduction, interprétation du contenu des DRSs, description d'un traitement en BACK,.... Par contre, en ce qui concerne la deuxième étape, seules sont données quelques indications générales sur le travail qu'il serait nécessaire d'accomplir pour construire un démonstrateur opérant sur des défauts.[2]

8.2. LA LOGIQUE DES DEFAUTS

Notre présentation de la logique des défauts sera intuitive. Pour une approche plus formelle et détaillée on peut se reporter à [REIT 80]. Les articles de référence sur les principales logiques non-monotones utilisées en IA sont rassemblés dans [RNNR 87]

Nous partons de l'idée que, dans le domaine des spécifications d'alarme, les connaissances sont mieux exprimées par des assertions de typicalité plutôt que par des lois universelles (cette assertion sera justifiée et détaillée dans la partie suivante):

```
(1) "Si la vitesse-sol est inférieure à Vmini, l'info est positionnée à
alarme"
(la vitesse-sol peut être invalide, l'info peut être forcée ...)
```

Cette hypothèse peut être étendue à des domaines plus généraux. Pour reprendre un exemple classique en Intelligence Artificielle :

```
(2) Les oiseaux peuvent voler
(sauf les autruches, les pingouins,...)
```

Nous savons donc, d'après le deuxième exemple, qu'en l'absence d'information contraire, les oiseaux volent. Le fait d'être un oiseau n'implique pas strictement le fait de pouvoir voler car nous n'avons pas une connaissance complète et définitive du problème. Il peut y avoir des exceptions. Certaines sont d'ailleurs mentionnées ici explicitement (autruches, pingouins). Cette connaissance peut se représenter par la règle de défaut suivante:

---

[2] Les idées avancées dans ce chapitre n'ont pas été implantées. Il s'agit donc de spécifications ou guides pour la deuxième phase du projet ACTES. La partie implantée correspond précisément à la première phase, validée par le ministère de la Défense et AMD-BA. L'implantation de la deuxième phase représente tout un travail en soi, au moins aussi important que celui de la première. Cela sortait donc du cadre de ce qu'il était pratiquement possible de réaliser dans les délais impartis. D'autant que les problèmes pratiques d'implantation ont été nombreux et ont retardé le projet de plusieurs mois: changement de langages Prolog (non compatibles entre eux) ; changement de versions dans un même Prolog (variantes parfois importantes), et dans BACK ; bugs (dans les Prolog et BACK), qui, certaines fois, se sont révélés graves et ont bloqué le développement ; changement d'ordinateurs (mini, micro puis à nouveau mini) ; disponibilité tardive d'une station graphique, bugs dans le serveur graphique, ...

$$(3)\ \frac{\text{oiseaux}(x) : \neg(\ \text{autruche}(x) \vee \text{pingouin}(x) \vee \ldots)}{\text{vole}(x)}$$

Cette règle se lit: si x est un oiseau et s'il ne peut pas être prouvé que x est un pingouin ou une autruche,..., alors on peut déduire que x vole. Donc avec la formule 'oiseau(tweety)' on peut inférer 'vole(tweety)'. Mais par contre, si on a également la formule 'autruche(tweety)', le défaut (3) n'est plus applicable, et 'vole(tweety)' n'est plus déductible. Nous avons ici un exemple de raisonnement non-monotone: une assertion nouvelle invalide des faits préalablement dérivés.

Plus généralement , une théorie des défauts est définie par un ensemble W de formules du premier ordre et un ensemble de défauts D. Un défaut est une expression de la forme:

$$\frac{u(x) : v(x)}{w(x)}$$

qui signifie grosso modo : si les conditions u(x) sont vérifiées et si v(x) est consistant (c'est à dire si $\neg$ v(x) n'est pas un théorème), alors il est possible d'inférer w(x). u(x) est le **prérequis** du défaut, v(x) sa **justification** et w(x) son **conséquent**.

L'ensemble des théorèmes de W est ainsi complété par l'application des défauts D et forme alors une **extension** de la théorie. Mais ces extensions peuvent ne pas être uniques ou ne pas exister. Différentes extensions de la théorie représentent différentes possibilités de compléter W en utilisant les défauts D.

Dans le défaut (3), les exceptions sont données en extension, ce qui suppose une connaissance complète du problème. Cette hypothèse est peu réaliste. En particulier le savoir que l'on a sur un texte en cours d'analyse est incomplet tant que l'analyse n'est pas terminée.

Ainsi, à partir de la connaissance "les oiseaux volent", nous pouvons construire un défaut semi-normal[3] :

$$(4)\ \frac{\text{oiseau}(x) : R1(x) \wedge \text{vole}(x)}{\text{vole}(x)}$$

Puis, pour traduire la connaissance supplémentaire "les autruches ne volent pas", il suffit d'ajouter le défaut:

[3] un défaut semi-normal a la forme suivante : (A : B $\wedge$ C) / C. B est appelé le **passant**.

$$(5)\ \frac{\text{autruche}(x) : R2(x) \wedge \neg R1(x)}{\neg R1(x)}$$

Ce défaut bloque le précédent. R2 est un nouveau **passant** introduit si l'on veut laisser la possibilité d'avoir une exception sur une exception. Si une nouvelle exception concernant le vol des oiseaux est connu, il suffira de rajouter un nouveau défaut. Le premier défaut (4) n'a pas besoin d'être réécrit. (5) sera appelé **règle d'annulation**. Cette utilisation du passant et des règles d'annulation a été introduite, en particulier, dans [FROI 88] pour représenter les exceptions dans les réseaux sémantiques.

8.3. APPLICATION AU PROBLÈME DES SPÉCIFICATIONS

Dans cette partie nous donnons une idée de ce que pourrait faire un expert en spécifications avec le système ACTES étendu[4] (c'est-à-dire un système incorporant un démonstrateur de logique des défauts), nous formalisons la notion de traitement correspondant au chapitre d'un texte, explicitons à partir de l'exemple du texte de reférence légèrement modifié les règles de défaut qui pourraient être produites et discutons enfin des avantages de notre démarche par rapport à la génération directe de règles de production.

8.3.1. Objectifs de l'expert

Le système expert en cours de développement à la société AMD-BA opère une **vérification de cohérence globale** sur l'ensemble des spécifications contenues dans les **tous les textes**: étant donné une état de l'ensemble des paramètres mesurés par les processeurs embarqués sur l'avion, quelles alarmes seront déclenchées à un instant précis ? dans quel ordre seront-elles déclenchées ? Cette simulation a pour but d'éviter une profusion de messages d'alarme sur le tableau de bord du pilote qui risquerait de lui masquer les informations les plus pertinentes.

Le système ACTES étendu peut opérer, à notre avis, à un niveau plus élémentaire : étant donné **un texte**, quelles règles peuvent en être extraites et quelles **vérifications de cohérence locale** à ce texte peuvent être effectuées ?

Un scénario d'utilisation du système par l'expert (cf figure 8.1) peut se décomposer en trois phases. Nous noterons T la théorie des défauts correspondant à un traitement. T =(W,D), avec W ensembles des axiomes et D ensemble des défauts. Nous assimilons une implication stricte à un cas particulier de défaut:

---

[4] Les objectifs que nous supposons être ceux de l'expert n'ont jamais été décrits précisemment dans les spécifications initiales du projet ACTES.

- **Phase 1, Description du traitement**: après avoir décrit un traitement sous forme de texte (un traitement correspond à un chapitre d'un texte, cf infra), l'expert le présente au système qui le traduit sous forme de DRSs puis de règles de défaut. A la fin de la phase 1, D est déterminé.

  Pendant l'analyse syntaxico-sémantique d'une phrase, le système vérifie certaines contraintes sémantiques entre groupes sujet et groupes verbaux (cf chapitre 6). D'autres contraintes, au niveau de la phrase sont vérifiées au moment de la traduction en règles de défaut (cf infra, la partie traduction DRSs-défauts). Le module de résolution d'anaphores vérifie une partie des relations entre les phrases du texte.

- **Phase 2, Description d'une situation** : l'expert fixe une valeur aux différents paramètres du traitement et décrit l'état des informations concernées. Le système traduit cette description sous forme d'axiomes de la théorie (W est fixé). L'expert peut alors poser des questions au système. Le démonstrateur de logique des défauts opère des déductions sur T (cf infra, la partie déduction en logique des défauts).

  Si les réponses du système ne correspondent pas à l'attente de l'expert, celui peut affiner sa description du traitement, ce qui se traduira généralement par la production de nouveaux défauts (cf phase 1). Il y a alors mise à jour de D.

- **Phase 3, Génération des règles pour le système expert**: L'expert jugeant la cohérence de son texte satisfaisante déclenche la traduction des règles de défaut D en règles de production destinées au système expert (cf infra, la partie traduction défauts-règles de production).

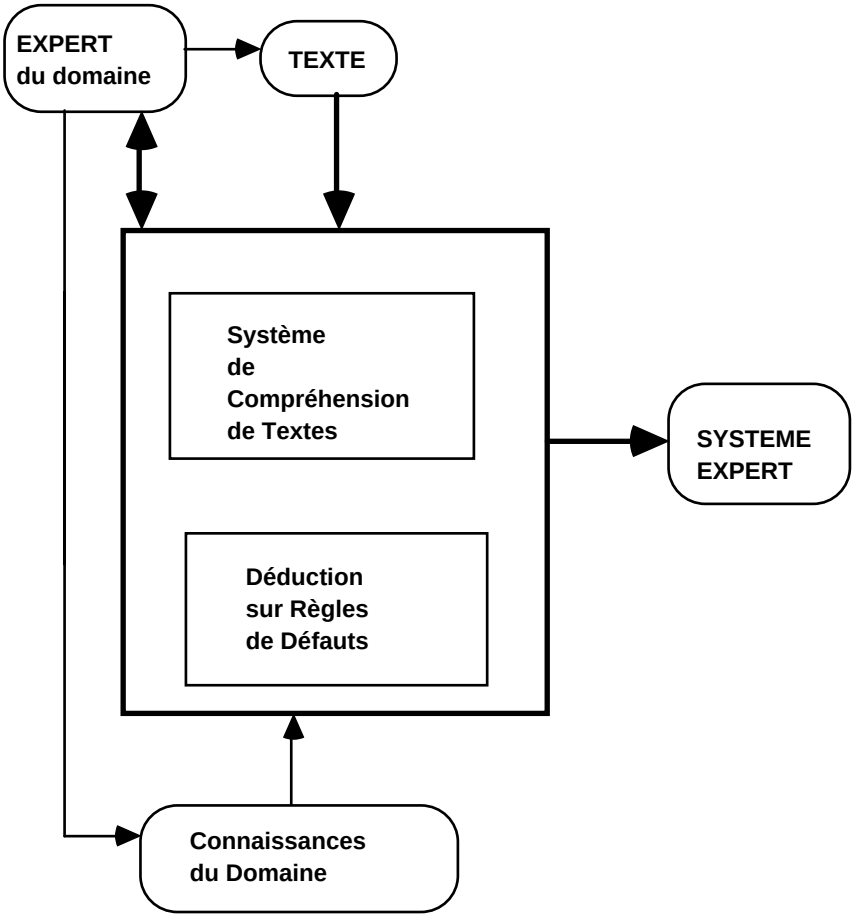


Fig 8.1: Système ACTES étendu avec vérification de cohérence locale

8.3.2. Forme générale d'un traitement

En phase 1, l'expert décrit un traitement particulier, ou plus exactement une instance d'une forme générale de traitement. Car l'étude des textes du domaine et les discussions avec les experts nous ont permis de caractériser, d'une part, un schéma général de traitement, d'autre part, une façon générale d'énoncer un traitement particulier. Nous revenons d'abord sur la notion de traitement évoquée au chapitre 5 en la caractérisant plus précisément.

8.3.2.1. Qu'est-ce qu'un traitement ?

L'ensemble de l'application (déclenchement d'alarmes sur un avion) se divise en plusieurs sous-pavés, chacun portant sur un sous-ensemble de l'avion (ex : le sous-pavé MOTEUR, le sous-pavé TRAIN d'atterrissage). Les textes correspondants à un sous-pavé sont structurés en chapitres. Chaque chapitre correspond à un traitement. L'ordre des chapitres n'est pas indifférent, il correspond à l'ordre séquentiel d'exécution des traitements.

Un traitement correspond à la notion de programme informatique. Les objets sur lesquels porte le traitement sont des informations. Le but du traitement (cf figure 8.2) est de décrire les règles positionnant une(ou conjonction) info(s) de sortie donnée(s) à alarme (ou non alarme). On appelera cette info, info de sortie principale. Eventuellement d'autres infos de sortie annexes pourront prendre la même valeur.

Une règle est composé d'une partie CONDITIONS (conjonction de conditions portant sur les valeurs d'infos d'entrée) et d'une partie ACTION (détermination valeur d'une info de sortie)[5]. Toute information a une valeur et pour certaines une validité.

Fig 8.2 : Description schématique d'un traitement

8.3.2.2. Structure du traitement

Un traitement est composé des parties suivantes :

* **contexte** (CONTEXTE): il est composé du nom du sous-pavé auquel appartient le traitement, de l'objet principal (infos de sortie principales), de l'objet secondaire (infos de sortie annexes), du but du traitement.
* **conditions d'application** (FORCAGE) : plusieurs règles (en fonction de l'invalidité de certaines infos d'entrée) forcent l'info de sortie principale à alarme (ou non alarme). La suite du traitement n'est alors plus applicable (REGLE-PRINCIPALE).
* **conditions générales** (REGLE-PRINCIPALE) : c'est le corps du traitement proprement dit. Il se subdivise en deux parties :

  - ensemble de CONDITIONS (éventuellement complexes) suivi de l'ACTION positionnant l'info principale à une valeur.

  - négation des CONDITIONS précédentes (cf "dans les autres cas") et ACTION positionnant l'info principale à la valeur contraposée.
* **effets de bord** (EFFET-DE-BORD) : ACTION(S) précisant quelles autres infos de sorties seront positionnées à une valeur identique à celle de l'info principale.
* **contrôle d'infos** (CONTROLES-SUR-ALARME) : cette partie est un sous-ensemble de REGLE-PRINCIPALE. En fonction de la validité de certaines infos d'entrée on change tout ou partie des CONDITIONS de cette partie.

[5] Nous donnerons plus loin une définition précise de la notion de règle.

La figure 8.3 décrit la structure d'un traitement en BACK. Cette structure sera instanciée par l'analyse du texte correspondant au traitement. Chaque règle de défaut figurera comme instance de certains des concepts du réseau. La partie CONTROLES-SUR-ALARME ne figure pas à ce niveau de description du réseau. Nous expliquerons tout cela de façon détaillée dans la partie traduction DRSs-Défauts.

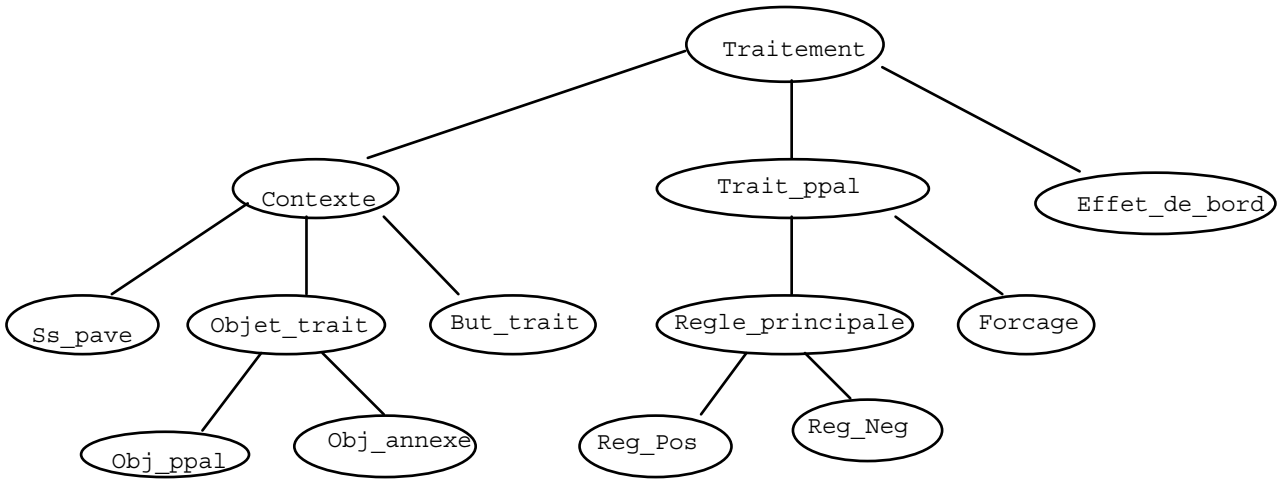


Fig 8.3: Description de la structure d'un traitement dans le réseau BACK

Tous les liens entre les concepts sont du type *partie_de*. En principe la relation entre un objet et ses partie se décrit, en BACK, au moyen d'un concept intermédiaire représentant l'ensemble des parties. Chaque partie est alors une instance de ce concept (cf figure 8.4). Notre dessin est donc une simplification.

8.3.3. Correspondance texte-traitement sur un exemple

En étudiant toute une série de textes du domaine, nous avons défini la structure générale d'un texte. Un texte est composé d'un titre et de sous-parties: REGLE_PRINCIPALE, CONTROLES_SUR_ALARME, ...(cf figure 8.4) Cette présentation est plus claire que l'utilisation un peu indifférenciée des divers notas et remarques présents dans les textes d'origine (cf figure 5.1). Elle est en relation directe avec la structure d'un traitement.

```
 1. Les infos de sortie:

 REGLE-PRINCIPALE :
             Alarme-gear-not-down et Disc-gear-not-down sont positionnées à
alarme si simultanément :
             - train-avant-verrouillé-bas est non-verrouillé-bas
             - train-gauche-verrouillé-bas est non-verrouillé-bas
             - train-droit-verrouillé-bas est non-verrouillé-bas
             - vitesse-conventionnelle est inférieure à Vmini
             - angle-manette-gauche est inférieur à angle-manette-mini
             - angle-manette-droite est inférieur à angle-manette-mini
             - hauteur-radio-sonde est inférieure à HRS-mini.

              Dans les autres cas, les sorties sont positionnées à non-
alarme.

 CONTROLES-SUR-ALARME:
              Si la vitesse-conventionnelle est invalide, on utilise la
vitesse-sol. Si la vitesse-sol est invalide également, on considère que la
condition sur la vitesse est remplie, c'est-à-dire inférieure à Vmini.
              Si la hauteur-radio-sonde est invalide, on utilise l'altitude-
baro-inertielle. Si l'altitude-baro-inertielle est invalide également, on
considère que la condition sur la hauteur est remplie, c'est-à-dire
inférieure à HRS-mini.
             Si un paramètre est invalide, on considère que la condition sur
le paramètre est remplie.

 FORCAGE:
Les deux infos de sortie sont forcées à non-alarme si l'info d'entrée, Val-
infos-TT-redondées, est à invalide.

 EFFET-DE-BORD:
On positionne l'info interne Train-non-sorti simultanément et à la même
valeur que les deux infos de sortie.
```

Fig 8.4: Texte du prototype modifié

Ainsi le texte de référence est sémantiquement composé de quatre parties: la première dans laquelle on énonce la règle principale, objet du traitement; la seconde dans laquelle on met des exceptions sur certaines informations intervenant en prémisse de la règle principale, lorsque ces informations sont invalides; la troisième où l'on bloque l'application de la règle principale: c'est le cas du forçage; enfin la dernière où l'on précise que tout le traitement précédent s'applique aussi pour une certaine information interne.

Mais la structure d'un texte et la structure d'un traitement ne sont pas identiques. Bien que nous ayons donné ici les mêmes noms à des parties d'un traitement et des parties d'un texte (pour limiter les notations dans notre exposé), les descriptions réelles dans BACK doivent être différentes. Il convient de ne pas confondre l'information textuelle avec celle liée au traitement parce que:

- les **données** contenues dans chacune des structures diffèrent. Après l'analyse d'un texte , les feuilles du réseau partiel décrivant sa structure textuelle contiendront des pointeurs vers les DRSs construites pendant l'analyse, tandis que les feuilles du réseau correspondant à la structure du traitement contiendront des informations sur les règles de défaut.

- l'**ordre** de description des deux structures n'est pas le même. Un traitement ne s'énonce pas comme il se définit. Ainsi la partie CONTEXTE du traitement est renseignée, dans l'exemple de la figure 8.4, par le titre du texte et les parties REGLE_PRINCIPALE et EFFET_DE_BORD.

La figure 8.5 donne une idée de la façon dont on peut décrire la structure d'un texte et sa relation avec la description d'un traitement en BACK. Rappelons que cette première description est nécessaire au module de découpage et au module de résolution des anaphores.

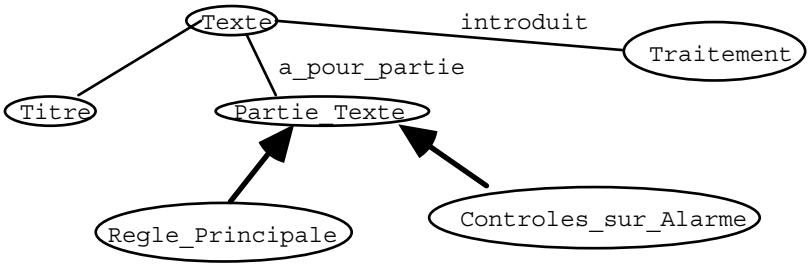


Fig 8.5: Description partielle d'un texte

Tous les noms de rôles ne sont pas explicités et leur notation est simplifiée par rapport à celle de la figure 6.4. Les flèches correspondent toujours au lien EST-UN.

8.3.4. Traduction partielle de l'exemple en défauts

Nous présenterons de façon intuitive comment sont construits les défauts correspondants au texte de la figure 8.4 et de quelle façon certains mots de la langue sont traduits logiquement.

La **première partie de la règle principale** correspondant à la première phrase peut se traduire par les défauts suivants:

$$(7) \quad \frac{C'_1 \wedge C'_2 \wedge C'_3 \wedge C'_4 \wedge \ldots \wedge C'_7 : R_1 \wedge A_1}{A_1}$$

avec A_1 : Alarme-gear-not-down et Disc-gear-not-down positionnées à non alarme

Chacune des infos intervenant dans une condition C'i de la première phrase peut être invalide (cf partie CONTROLES_SUR_ALARME). Il faut donc générer un défaut par condition, avec un passant Ri différent à chaque fois. Si l'info n'est pas un booléen ou est un paramètre principal alors le défaut aura la forme suivante:

$$(7b)\qquad \frac{: R_i \wedge C'_i}{C'_i}$$

Si l'info est un paramètre secondaire, alors le défaut (cf (8)) devra avoir pour conséquent C'i, obtenu à partir de la condition Ci de la phrase en remplaçant le paramètre secondaire par le paramètre principal.

$$(8)\qquad \frac{C_4 : R_2 \wedge C'_4}{C'_4}$$

avec C'_4 : vitesse inférieure à Vmini
et C_4 : vitesse conventionnelle inférieure à Vmini.

Expliquons-nous. La valeur de certains paramètres, appelés paramètre principaux, comme la vitesse et la hauteur, peut être déterminée par la lecture de la valeur de plusieurs paramètres, dits secondaires, comme la vitesse-sol, la vitesse-conventionnelle (pour la vitesse), ou la hauteur-radio-sonde et l'altitude-baro-intertielle (pour la hauteur). Dans (7) ne doit donc figurer que les conditions portant sur les paramètres principaux (ou sur des booléens).

La figure 8.6 représente quelques relations entre paramètres principaux et paramètres secondaires.

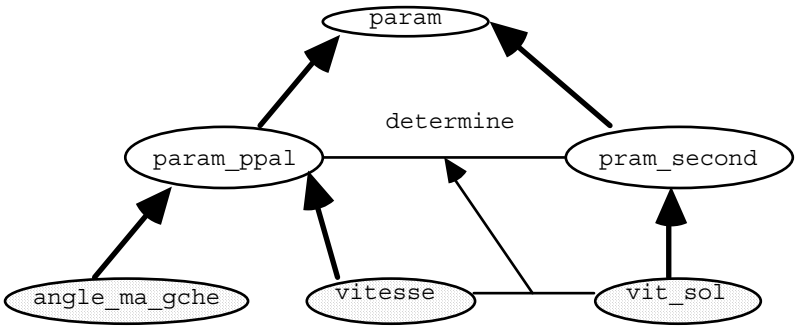


Fig 8.6: Relations entre paramètres principaux et pramètres secondaires

La **deuxième partie de la règle principale** ("Dans les autres cas,...") se traduit:

$$(9)\qquad \frac{\neg\ (\ C'_1 \wedge \ldots \wedge C'_7)\ :\ R_1 \wedge \neg\ A_1}{\neg\ A_1}$$

Le passant R_1 est le même que dans le premier défaut (7) de façon à bloquer toute la règle principale, comme nous allons le voir.

La **troisième partie sur le forçage**, lorsque val-info-tt-redondées est invalide (condition C_8), se traduit :

$$(10)\qquad \frac{C_8\ :\ R_3 \wedge \neg\ R_1}{\neg\ R_1}\qquad \text{pour bloquer les défauts (7) et (9)}$$

et le verbe "forcer" se traduit par une implication stricte[6]:

$$(11)\qquad C_8 \rightarrow \neg\ A_1 .$$

La deuxième partie du texte concernant les cas d'invalidité pour les informations présentes en prémisses de la règle principale sera traduite par une succession de défauts emboîtés.

Par exemple, la phrase où la vitesse-conventionnelle est supposée invalide se traduit:

$$(12a)\qquad \frac{C_9\ :\ R_4 \wedge \neg\ R_2}{\neg\ R_2}$$

et

$$(12b)\qquad \frac{C_9 \wedge C''_4\ :\ R_4 \wedge C'_4}{C'_4}$$

avec C_9: vitesse-conventionnelle invalide
C''_4 : vitesse-sol inférieure à Vmini.

Celle où la vitesse-sol est invalide et où l'on considère la condition comme remplie:

---

[6] En "bonne logique", (11) suffit à bloquer (7) et il ne sert à rien de bloquer (9). Mais ce que nous exprimons en mettant le même passant dans (7) et (9), c'est que ces deux règles sont solidaires, (quel qu'en soit leur contenu respectif) et que (11) les bloquera toutes les deux. Ceci correspond au fait que dans un traitement les conditions d'application sont considérées avant les conditions générales. Nous retrouverons cette notion d'ordre sous-jacente dans la partie traduction Défauts-règles de production : les implications strictes seront examinées en premier.

$$(13) \qquad \frac{C_9 \wedge C_{10} : R_5 \wedge \neg R_4}{\neg R_4}$$

le verbe "considérer" donne une implication stricte:

$$(14) \qquad C_9 \wedge C_{10} \rightarrow C'_4$$

8.3.5. Régles de défaut et règles de production

La partie précédente montrait la bonne représentation du processus d'extraction des règles à partir du texte qu'offre la logique des défauts. De façon plus générale le processus d'analyse d'un texte est lui aussi non-monotone.

L'analyse de texte est séquentielle: les phrases sont analysées une à une dans l'ordre du texte. Supposons que les phrases du texte soient des affirmatives et que chaque phrase analysée soit traduite en une formule logique. Si l'analyseur est connecté à une BD et qu'une phrase exprime un fait, il est fréquent de traduire cette assertion par une mise à jour immédiate de la BD. Or une phrase suivante peut invalider en tout ou partie ce qui vient d'être dit. Il est nécessaire alors de défaire l'assertion accomplie au pas précédent et de modifier la formule logique correspondante.

Ce travail peut se répéter un nombre indéfini de fois car, souvent, aucune indication n'est donnée dans le contenu d'une phrase sur la portée de l'assertion correspondante. Face à ce problème on aimerait disposer d'un formalisme méta-logique pouvant bloquer les inférences contenues dans les formules logiques produites, représentant et gérant les exceptions au fur et à mesure qu'elles se présentent.

C'est précisément cela que permet la logique des défauts. La construction des défauts est séquentielle, comme l'analyse du texte. Chaque phrase donne lieu à la construction d'au moins un défaut. Pour représenter, à priori, la possibilité de remise en cause de ce qui vient d'être construit, on place systématiquement un passant R dans le défaut *d* produit. Cette hypothèse est minimale, elle ne préjuge pas du contenu de la suite du texte. Si aucune phrase, par la suite, ne vient invalider ce qui a été analysé, alors aucune règle de défaut ne conclura sur la négation de R. *d* pourra s'interpréter comme une formule logique traditionnelle. Dans le cas contraire, on peut bloquer les inférences correspondantes à la première phrase sans réécrire *d*.

Cette approche est suffisamment souple pour tenir compte également de certains éléments du langage donnant une indication de la portée d'une assertion contenue dans la phrase. Dans l'exemple du texte de référence, les verbes "forcer" et "considérer" peuvent s'interpréter comme introduisant une assertion qui ne pourra être remise en cause. Aucun justificatif ne sera alors introduit et ces phrases se traduiront en implicatives strictes (implicatives de la logique traditionnelle).

Enfin, avec la logique des défauts nous disposons d'un cadre pour la vérification de cohérence d'un texte de spécification et la construction d'un système interactif d'élaboration d'une spécification. La réponse à une question du genre "Si je suis dans tel état, que se passe-t-il ?" s'obtient en construisant la ou les extension(s) correspondante(s) à la théorie (W,D) obtenue par l'analyse du texte précédent cette question. L'utilisateur peut ajouter de nouvelles spécifications et vérifier incrémentalement leur cohérence (cf partie suivante).

C'est donc pour toutes ces raisons que nous proposons de produire des règles de défaut à partir de la représentation sémantique intermédiaire fournie par l'analyseur de texte et d'opérer la déduction sur ce formalisme. La génération de règles de production à partir de défauts ne se ferait qu'en dernière phase, lorsque l'utilisateur indiquerait au système qu'il juge son texte définitif et suffisamment cohérent après l'avoir construit incrémentalement en dialoguant avec le système.

8.4. EXTENSION DE ACTES AVEC LES DEFAUTS

Nous présentons quelques-uns des axes de recherche à développer pour étendre le système ACTES avec la logique des défauts et construire un système interactif de vérification de cohérence. Trois étapes sont envisagées : production des défauts à partir de la représentation sémantique intermédiaire d'un texte, déduction sur les défauts, traduction des défauts en règles de production. Chaque étape soulève des problèmes de nature très différente et ne sera pas abordée avec le même niveau de détail.

La traduction DRSs-Défauts, par exemple, est un problème de traitement du langage naturel: une règle de défaut est vue comme une structure de données et le programme de traduction doit, en quelque sorte, interpréter de façon procédurale le contenu des DRSs pour produire de telles structures de données. Un algorithme de traduction sera donnée pour l'exemple du texte de référence.

La déduction sur les défauts relève elle des problèmes de démonstration automatique. La théorie des défauts a la désagréable propriété de n'être pas semi-décidable. Il n'est donc pas possible de construire un programme qui prouve systématiquement en un temps fini qu'une formule appartient bien à une extension. Rappelons que la logique des prédicats du premier ordre est elle semi-décidable (mais pas décidable). L'idée est donc de caractériser un sous-ensemble syntaxique des défauts suffisant pour représenter les textes du domaine et qui serait semi-décidable. Cette étape, au contraire, de la précédente ne sera considérée que d'un point de vue très général.

Enfin la troisième phase, traduction Défauts- règles de production n'est abordée que de façon intuitive et sur un exemple limité. Elle indique une partie des règles de production qui pourraient être produites à partir des défauts extraits du texte de référence et explicités partiellement au paragraphe 8.3.5.

8.4.1. Traduction DRSs - Défauts sur un exemple

Le problème de la traduction des DRSs en défauts est abordé à partir de l'exemple du texte de la figure 8.4. Le contenu des DRSs produites à partir de ce texte figure en annexe. Trois points vont être évoqués:

- la construction des défauts ne peut s'effectuer qu'à partir de DRSs complètement instanciées. Il va donc falloir mettre en évidence certaines informations implicites.
- la structure d'une règle de défaut doit être décrite dans BACK. Cette description, ainsi que celles du traitement vont être utilisée pour construire les défauts. Mais elles peuvent aussi servir à effectuer, en même temps, des vérifications de cohérence de portée limitée à la phrase.
- l'algorithme de construction des défauts est donné. Nous proposons une interprétation procédurale de certaines conditions contenues dans les DRSs.

8.4.1.1. Informations implicites

Après l'analyse syntaxico-sémantique du texte de référence, les arguments de certains des littéraux apparaissant dans les conditions des DRSs ne sont pas instanciés. Cela correspond au fait que des informations implicites du texte n'ont pas été mises en évidence. Nous indiquons comment les expliciter en utilisant le module de résolution d'anaphores présenté au chapitre 7.

Dans les phrases (ph1) et (ph2) le texte ne précise pas l'endroit où doivent être utilisées les informations de type *vitesse* et *hauteur*.

(ph1) "si la vitesse-conventionnelle est invalide, on utilise la vitesse-sol. (*dans la condition sur la vitesse*)"

(ph2) "si la hauteur-radio-sonde est invalide, on utilise l'altitude baro-intertielle. (*dans la condition sur la hauteur*)"

La grammaire précise cependant qu'une information d'entrée doit être utilisée dans une condition:

```
utilise:
<cat> = verbe
<pred> = utilise(X1,X2)
<arg1> = X1:entree
<arg2> = X2:condition
```

Le littéral introduit par le groupe verbal de (ph1) dans la DRS a cette forme:

```
utilise(vit_sol,X2)
```

Il s'agit donc de déterminer X2 sachant que X2 est du type *condition*. Rappelons que *utilise* est un concept du réseau qui est en relation avec les concepts *entree* et *condition* (cf chapitre 6). Le problème peut alors s'énoncer en disant que ces deux rôles sont essentiels et que leur portée doit être définie.

L'idée est de marquer le rôle comme 'essentiel' en ajoutant une nouvelle information à la variable correspondante (cf chapitre 7). Lors de l'analyse du verbe, cette variable est introduite sous forme de Dref dans l'univers de la DRS courante. Si, à la fin de l'analyse, ce Dref n'a pas été instancié alors, il sera considéré comme un terme anaphorique et le module de résolution d'anaphores cherchera à trouver la condition correspondante. Dans le cas de l'exemple (1), il s'agit de la condition sur la vitesse de la règle principale, c'est à dire la condition "si la vitesse-conventionnelle est inférieure à Vmini" qui se traduit *inf(vit-conv, vmini)* (cf en annexe, la recherche de l'antécédent pour "la condition sur la vitesse est remplie").

Notre façon de traiter le problème des constituants sémantiques implicites en s'appuyant sur le module de résolution d'anaphores est similaire à celle choisie dans le système de compréhension de textes PUNDIT [PALM 86] pour la recherche des informations implicites.

8.4.1.2. Structure d'une règle, vérifications locales

Nous décrivons schématiquement la structure d'une règle en BACK et indiquons quels usages peuvent être faits de cette description et de la description du traitement donnée précédemment (cf figure 8.3).

Une règle (cf figure 8.7) a un numéro d'identification, des prémisses que nous appelerons prérequis ou cas (cf chapitre 7) et un conséquent. Il existe deux types de règles les défauts, qui ont un passant, et les implications strictes[7].

[7] Nous avons présenté précedemment les implications strictes comme des cas particuliers de défauts. Ici nous les distinguons puisqu'un défaut a un passant.

Un conséquent sera un concept de type *action* comme *considerer*, *forcer*, *positionner* ou le concept *utiliser*. Tous ces concepts ont deux rôles et sont traduits dans les conditions des DRSs par des prédicats binaires comme *utiliser(Info,Condition).* Un cas est un ensemble de conditions élémentaires. Une condition élémentaire est une comparaison (cf *inf(vit_conv, vmini)*) ou un état (cf *etre(vit_sol, invalide)*). Dans les conditions élémentaires, certaines se rapportent à des paramètres (*cond_elem_param*) et ces dernières se rapportent soit à des paramètres principaux (*cond_elem_param_ppal*) soit à des paramètres secondaires (*cond_elem_param_second*). Le lecteur peut consulter l'annexe pour une description plus complète des conséquents et des conditions élémentaires.

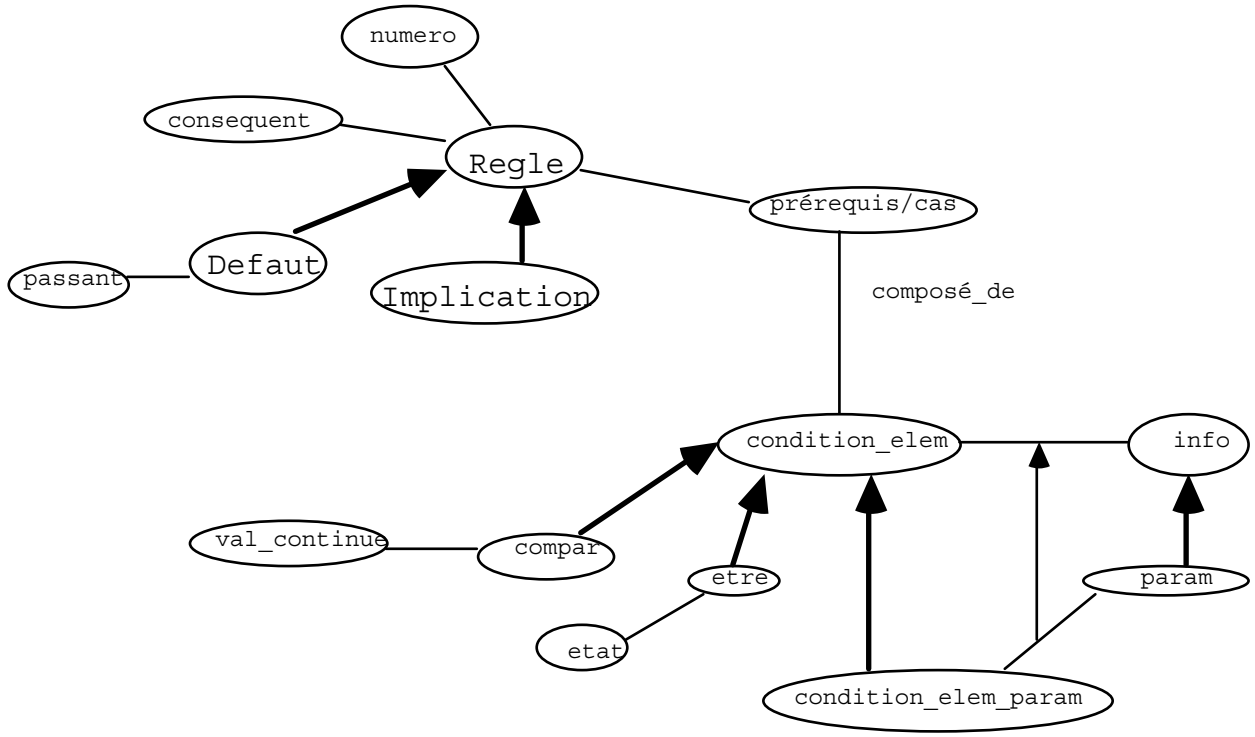


Fig 8.7: Représentation partielle d'une règle et d'une condition élémentaire

A quoi sert cette description du traitement et des règles ? Les phrases qui sont des conditionnelles dans le texte (toutes sauf la dernière, dans la partie EFFET-DE-BORD), sont traduites par l'analyse syntaxico-sémantique en des DRSs dont les conditions sont des conditions implicatives: une DRS antécédente et une DRS conséquente. Chaque DRS donnera lieu à la production d'au moins une règle qui sera assertée à la fois en mémoire centrale, afin d'être utilisée pour faire de la déduction, et dans BACK, pour que le programme de traduction DRSs-défauts puisse disposer des informations sur toutes les règles en cours de génération. En particulier les concepts *Reg_Pos*, *Reg_Neg* et *Forcage* (cf figure 8.3) auront des instances qui seront des règles de structure similaire à celle décrite dans la figure 8.7. La création de ces règles dans BACK se fera sous forme d'assertion dans la ABOX.

Ainsi, lorsque des exceptions seront analysées, comme dans la partie CONTROLES-SUR-ALARME, toutes les informations sur les règles créées antérieurement (et en particulier leur numéro et leur passant) pourront être retrouvées. BACK nous permettra, de plus, de faire les inférences nécessaires pour, par exemple, identifier les conditions élémentaires portant sur des paramètres secondaires ou principaux qui recevront un traitement particulier comme nous le verrons dans l'algorithme de traduction.

D'autre part, la description du contexte du traitement qui se trouve éclatée, comme nous l'avons dit, dans le texte sera complétée au fur et à mesure de la traduction.

Cette description du traitement et des règles peut également servir à **vérifier la cohérence d'une phrase**. Il est possible de traduire chaque phrase par une assertion dans la ABOX dont le moteur d'inférence vérifirait la cohérence. Pour ne prendre qu'un exemple trivial, si une phrase indique: "on positionne info1 à valeur1 et valeur2", BACK rejettera une telle assertion puisqu'une information ne peut être positionnée à un instant donné qu'à une et une seule valeur (cardinalité sur les rôles). Ces assertions dans la ABOX doivent être locales à la phrase en cours d'analyse, puisqu'il est parfaitement possible dans une phrase de parler du positionnement d'une information à une valeur, puis dans une phrase suivante du positionnement de cette même information à une autre valeur, sans que ces hypothèses soient contradictoires. BACK dispose de la notion de **contexte** [LUCK 86]. Des assertions contradictoires dans la ABOX sont tôlérées si elles ont lieu dans des contextes différents. On dispose donc d'un moyen aisé de limiter la portée des assertions.

Ces vérifications de cohérence locales à la phrase (que nous n'avons pas implantées jusqu'à présent) viendraient compléter utilement les vérifications élémentaires de typages opérées dans la grammaire[8] (cf chapitre 6).

8.4.1.3. Algorithme de traduction DRSs-Défauts

Le principe de la traduction des DRSs en défauts est, maintenant, explicité partie par partie. Le traitement d'une règle est détaillé plus particulièrement. C'est à ce niveau que les littéraux des DRSs sont "interprétés". Pour bien comprendre cet algorithme, il est nécessaire de se reporter aux exemples de défauts donnés précédemment en partie 8.3.4.

Nous utiliserons fréquemment les prédicats Prolog suivants:

- **generer_regle(Num, Type_regle, Prerequis, Passant, Consequent)**: assertion d'une règle en mémoire centrale et dans la structure BACK (assertion dans la ABOX). Le programme Prolog abrégé pourrait avoir la forme suivante (nous avons omis les arguments indiquant la partie en cours de traitement).

---

[8] Procéder à ces vérifications locales à chaque phrase ne peut se faire pendant l'analyse syntaxico-sémantique car cela ralentirait fortement cette analyse. C'est pour cette même raison que les contraintes de type vérifiées en cours d'analyse sont réalisées à partir d'information en provenance de BACK (cf chapitre 6), mais ne sont pas directement accomplies par le classificateur de BACK.

```
generer_regle(Num, implication,Prerequis, _, Consequent) :-
        !,
        assert( regle(Num, Prerequis => Consequent)),
        renseigner la structure dans BACK.

generer_regle(Num, defaut, Prerequis, Passant, Consequent):-
        assert( regle(Num, Prerequis, Passant, Consequent))
        renseigner la structure dans BACK.
```

- **cond_param_second(Cond, NCond)**: Si la condition élementaire *Cond* est une comparaison sur un paramètre secondaire alors substituer le paramètre secondaire par le paramètre principal dans *Cond*. On obtient alors *NCond*. La vérification est obtenue en faisant appel à BACK.
- **nouveau_passant(P)**: génère un nouveau passant, appel à BACK qui renvoie un concept individuel de la ABOX.
- **num_regle(N)**: génère un nouveau numéro pour une règle. Même principe que précédemment.

1) Titre

Si des informations de sortie sont présentes dans le titre, renseigner la partie *contexte* du réseau BACK. Ce sont les informations principales du traitement.

2) Regle-Principale

- 1° phrase correspondant à la partie *Reg_Pos* du réseau :
  * si les informations principales du contexte n'ont pas été précisées dans le titre alors faire le même traitement que dans 1).
  * faire *nouveau_passant(P1)* et appel de *traitement_regle(P1,_, Consequent, DRSante, DRScons)*. *DRSante* et *DRScons* sont respectivement les DRSs antécédente et conséquente de la phrase courante.
  * *Consequent* est le but du traitement. Renseigner la partie correspondante du *contexte* du réseau si cela n'a pas déjà été fait dans 1).
- 2° phrase correspondant à la partie *Reg_Neg* du réseau: appel de *traitement_regle(P1,_,_, DRSante1, DRScons1)*. *DRSante1* et *DRScons1* sont respectivement les DRSs antécédente et conséquente de la phrase courante. Le passant est le même que précédemment.

3) Controles-sur-alarme

- Sur chacune des phrases de la partie, faire:
  * *nouveau_passant(P)*
  * appel de *traitement_regle(P, _, _, DRSante, DRScons).*

4) Forcage

- Recherche du passant de la règle principale, P1, faire *nouveau_passant(P)* et *traitement_regle(P, non P1,_, Drsante, DRScons).*

5) Effet-de-bord

La phrase de cette partie ne décrit pas de règle, mais seulement une action à exécuter. Comme dans le cas des autres actions présentées dans la partie *traiter_consequent* nous ferons une interprétation procédurale des littéraux de la condition de la DRS correspondante.

Il s'agit de positionner une information (Info) à la même valeur que d'autres informations (Infos) et d'accomplir ce positionnement de façon simultanée pour toutes les informations, ce qui peut se traduire par la procédure : dans toutes les règles où les informations Infos sont positionnées, alors changer Infos par Infos + Info.

6) traitement_regle(Passant, Passant_Forc, Consequent, DRSante, DRScons)

La valeur du passant n'est pas déterminée dans *traitement_regle*, mais est connue à l'appel : deux règles générées à partir de deux phrases distinctes peuvent, en effet, avoir le même passant (cf (7) et (9)). Le type de la règle (défaut ou implication) est déterminé par le littéral, correspondant au verbe de la phrase, présent dans la DRS conséquente. Le prérequis peut être déterminé partiellement par le programme *traiter_consequent* et des défauts générés par le programme *traiter_cas*.

Pour illustrer ces deux derniers points, prenons l'exemple (12): *traiter_cas* produit (12a) et détermine C_9 ; t*raiter_consequent* determine $C"_4$. Donc le prérequis est la conjonction de C_9 et $C"_4$. (12b) est produite par *generer_regle*.

Faire:

- *num_regle(Num)*
- *traiter_cas(Passant, Passant_Forc, Cas, DRSante)*
- t*raiter consequent(Consequent, Type_Regle, Cas2, DRScons),*
- *Prerequis = Cas2 + Cas*
- *generer_regle(Num, Type_Regle, Passant, Conséquent, Prerequis).*

Traiter_consequent(Consequent, Type, Cas, DRScons)

Les littéraux de la DRS conséquente sont interprétés procéduralement. Ainsi "considerer une condition comme remplie" se traduira tout simplement par la fait que cette condition deviendra le conséquent de la règle à générer. Si l'information présente dans cette condition est un paramètre secondaire, il faut auparavant la remplacer par le paramètre principal correspondant.

On traite le contenu de *DRScons:*

- si:
  * *action = positionner* ou *utiliser* alors *Type = defaut*
  * *action = forcer* ou *considerer* alors *Type = implication*

- exécuter l'action:

* ***considerer****(Cond,rempli)* alors faire

cond_param_second(Cond, NCond)

Consequent = NCond

* ***positionner****(Info,Valeur) = Consequent*

* ***forcer****(Info, Valeur) = Consequent*

* ***utiliser****(Info,Cond)* alors faire

cond_param_second(Cond, NCond)

Consequent = NCond

Cas = la condition *NCond* dans lequel on remplace l'information par *Info*.

traiter_cas(Passant, Passant_Forc, Cas, DRSante)

Les DRSs antécédentes du texte prototype sont toutes composées d'une conjonction de conditions élémentaires[9].

- Traiter chacune des conditions élémentaires de *DRSante*.

Pour chaque condition élémentaire, il faut identifier sa nature à l'aide de BACK. Si cette condition, *Cond_elem* , est une :

* **invalidité** sur l'info *Info*, alors:

si *Passant_Forc* est instancié alors *Passant_Forc = P1*, sinon recherche dans BACK du passant *P1* du défaut dont le prérequis est un condition élémentaire portant sur *Info*.

faire :

num_regle(Num1)

generer_regle(Num1, defaut, Cond_elem, Passant, non(P1)). génération d'un défaut dont le conséquent est la négation de P1 ; le passant est celui qui a été instancié à l'appel de *traiter_cas*. (cf (10), (12a)).

* **comparaison sur un paramètre secondaire** (cf (8)) , alors faire:

cond_param_second(Cond_elem, NCond)

num_regle(Num1)

nouveau_passant(P1)

generer_regle(Num1, defaut, Cond_elem, P1, NCond).

* **autre comparaison** (cf (7b)), alors faire

---

[9] Dans certains textes, on peut trouver des conditions disjonctives. Le traitement correspondant deviendrait plus complexe: il faudrait d'abord modifier la représentation du concept *prerequis* dans la description d'une règle (figure 8.7), exécuter autant de fois *traiter_cas* qu'il y a de disjonctions. Plusieurs règles de défaut seraient générées, mais toutes avec le même passant.

num_regle(Num1)
nouveau_passant (P1)
generer_regle(Num1,defaut, [], P1, Cond_elem).

- *Cas* = union des conditions élémentaires

8.4.2. Déduction en logique des défauts

8.4.2.1. Rester en logique des propositions

Dans un premier temps, il serait raisonnable de se restreindre à la logique des propositions, et donc proscrire toute utilisation de variables logiques et quantificateurs dans les défauts. Cela éviterait les problèmes de skolémisation rencontrés par les équipes travaillant avec le langage des prédicats complet [CAST 88]. Cette hypothèse ne semble pas être trop contraignante car les textes étudiés sont assez précis et limités pour permettre de définir en extension les éléments des ensembles rencontrés et donc éviter tout recours aux quantificateurs existentiel et universel.

8.4.2.2. Garantir la non-circularité

Les défauts utilisés ici sont semi-normaux. Une condition suffisante pour qu'il existe au moins une extension, est que ces défauts soient ordonnés [ETHE 87][10]. Le cas où les défauts ne sont pas ordonnés correspond, dans les réseaux sémantiques, à l'existence d'un cycle. Un tel cycle existerait si, par exemple, on remplaçait (13) par:

$$(15) \qquad \frac{C_9 \wedge C_{10} : R_2 \wedge \neg R_4}{\neg R_4}$$

Il y aurait alors conflit avec le premier défaut de (12). Le problème vient du fait qu'on utilise deux fois le passant R2 dans une règle d'annulation. Outre le fait que le défaut (15) ne correspond à rien d'exprimable en rapport avec le texte, le problème est levé si on génère un passant nouveau dans chaque règle d'annulation.

Pour garantir la non-circularité, il est nécessaire de définir précisément l'alphabet du langage utilisé dans notre théorie des défauts, ainsi que les types de défauts (semi-normaux, règles d'annulation semi-normales, ...).

---

[10] L'utilisation de défauts normaux (A:B/B) garantirait l'existence et l'unicité d'une extension (si les défauts sont clos et l'union de l'ensemble des axiomes de la théorie et de l'ensemble des conséquents des défauts est consistant), mais leur utilisation pose des problèmes d'interactions entre défauts ([CAST 88]).

8.4.2.3. Preuves dans notre théorie des défauts

"Si je suis dans tel état que se passe-t-il ? ". Pour répondre à une question de ce genre, il faut que le système ait pris en compte les **faits décrivant l'état** (axiomes supplémentaires W qui devient W') et à partir de la théorie (W',D) **calcule toutes les extensions**. L'utilisateur pourrait alors voir si sa spécification n'est pas ambiguë (plusieurs extensions peuvent exister représentant chacune des situations contradictoires).

"**Si je suis dans tel état, est-il vrai que P ?"** Ici, on ne peut se contenter de trouver une séquences de preuves de défauts (au sens de [REIT 80]) concluant sur P, pour pouvoir affirmer P. P sera vrai s'il est dans au moins une extension. Il se peut en effet que lors du calcul d'une extension, une séquence de preuves nous amène dans une première étape à conclure sur P, puis, dans une étape ultérieure, à conclure sur ¬ P ou sur des faits mettant en cause les justificatifs qui nous ont permis de conclure sur P. Dans ce dernier cas, P doit être retiré de l'ensemble des faits établis.

Pour résoudre le problème précédent nous nous heurtons à deux difficultés sur le plan formel : pouvons nous garantir l'existence d'une extension (cf partie précédente) ? Avons nous une procédure de calcul répondant "oui" toutes les fois que P appartient à une extension (semi-décidabilité déjà évoquée) ? Même si la réponse est "non", dans le cas général, [ETHE 87] a montré qu'il existe des sous-ensembles de défauts semi-normaux sur lesquels on peut garantir la semi-décidabilité. Il nous faut donc caractériser un tel sous-ensemble et implanter les procédures de calcul des extensions.

Avant d'aborder succinctement le problème de l'implantation, il faut remarquer que ce travail peut commencer sans avoir la preuve formelle de semi-décidabilité. Ce point de vue a été défendu par [REIT 80] qui parlait de l'importance de l'**heuristique** dans le traitement des défauts dans une théorie non semi-décidable. Et dans un domaine proche de celui qui nous intéresse (comme nous allons le voir), le système de maintenance de la cohérence de [DOYL 78] a eu beaucoup d'émules alors qu'il n'avait pour toute preuve que celle contenue dans ses lignes de programmes.

Comment **implanter un système de preuves de défauts** ? Construire une preuve implique la **révision de croyances** : dès lors que de nouveaux faits peuvent venir en contredire d'autres déjà présents dans une base de données, la question se pose de l'élimination des conséquences des faits infirmés, pour conserver à la base sa fiabilité. Ce problème est abordé par [DEKL 86] qui a mis au point des algorithmes efficaces pour développer des inférences à partir de suppositions dans des contextes différents. Des faits contradictoires peuvent être présents simultanément s'ils ne sont pas dans le même environnement. Si une contradiction apparaît dans le même environnement, son système permet de retrouver le contexte présent avant l'inférence du premier fait concerné par la contradiction. Certains langages, comme KEE [FILM 88], ont permis d'implanter le système ATMS de De Kleer.

Il serait intéressant d'utiliser l'approche précédente pour construire un système de démonstration dans notre sous-ensemble des défauts. Cela nécessiterait certaines adaptations car nous devons gérer non seulement les faits contradictoires, mais également tout fait niant le justificatif d'un défaut utilisé dans la démonstration. Dans ce dernier cas, on ne peut conclure avec certitude à une contradiction.

En conclusion de cette partie, signalons la proximité de notre problématique avec celle de [CAST 88].

8.4.3. Traduction défauts - règles de production

Nous sommes dans la situation où l'utilisateur indique au système qu'il a terminé la vérification interactive de son texte de spécification. Il convient alors de traduire les défauts en règle de production pour le système expert.

Si aucun fait n'a été introduit suite à l'analyse du texte (nous ne parlons pas ici des faits introduits par l'utilisateur au moment de la vérification de la cohérence de son texte), il n'y a pas de preuve à faire et le problème reste sur le plan syntaxique: trouver les dépendances entre les règles à travers les passants.

Ainsi (16) dépend des règles d'annulation (17) et (18). Si aucune règle ne conclut ni sur $\neg R_j$ ni sur $\neg R_k$, (16) peut se réécrire en règle du premier ordre (19).[11]

$$(16)\qquad \frac{A : R_i \wedge C}{C}$$

$$(17)\qquad \frac{F : R_j \wedge \neg R_i}{\neg R_i}$$

$$(18)\qquad \frac{D : R_k \wedge \neg R_i}{\neg R_i}$$

$$(19)\qquad A \wedge \neg(D \vee F) \rightarrow C$$

En appliquant ce principe au texte, voyons comment construire les règles concluant sur la condition C'_4 : vitesse inférieure à Vmini.

---

[11] Sur le plan logique, il n'y pas équivalence entre ((16), (17), (18)) et (19). En particulier dans un cas, on peut conclure sur C en connaissant A seulement, alors que dans (19), ce n'est pas possible. La non-équivalence se justifie par le fait que les situations ne sont pas les mêmes dans les deux cas: dans la phase 1 du scénario, la description du traitement n'est pas définitive, de nouvelles exceptions peuvent toujours être ajoutées par l'expert ; dans la phase 3 de génération des règles de système expert, on considère que tout a été dit sur le traitement (cf hypothèse du monde fermé). Dans ce dernier cas, on connaît toutes les exceptions à l'application d'une règle.

Les défauts et règles concernés sont (8), (12b) et (14).

(8) $$\frac{C_4 : R_2 \wedge C'_4}{C'_4}$$

(12 a) $$\frac{C_9 : R_4 \wedge \neg R_2}{\neg R_2}$$

(12 b) $$\frac{C_9 \wedge C''_4 : R_4 \wedge C'_4}{C'_4}$$

(13) $$\frac{C_9 \wedge C_{10} : R_5 \wedge \neg R_4}{\neg R_4}$$

(14) $$C_9 \wedge C_{10} \rightarrow C'_4$$

En partant de l'implication stricte (14), il est possible de parcourir les dépendances à travers (13), (12a) et (8), ce qui nous donnera les implications suivantes:

$$C_9 \wedge C_{10} \rightarrow C'_4$$
$$C_9 \wedge C"_4 \wedge \neg C_{10} \rightarrow C'_4$$
$$C_4 \wedge \neg C_9 \rightarrow C'_4$$

8.5. CONCLUSION

Des problèmes de natures très différentes ont été évoqués dans ce chapitre avec des niveaux de détails eux-aussi fort différents suivant les cas.

Un des objectifs du chapitre était de montrer comment nous interprétions les DRSs produites par l'analyse syntaxico-sémantique. Cela ne pouvait se faire sans présenter, au préalable, la logique des défauts. Ce processus d'interprétation est relativement bien détaillé puisqu'il est abordé à partir d'un exemple et réfère à des parties du système ACTES qui ont été implantées: module de résolution d'anaphores, connexion avec BACK,... Dans cette partie, nous avons montré comment utiliser toutes les potentialités de BACK: TBOX, ABOX, classificateur.

Le deuxième objectif était de présenter les avantages d'une représentation intermédiaire des règles extraites d'un texte dans le formalisme de la logique des défauts et donner des indications succinctes sur les tâches à accomplir dans une phase ultérieure de développement du projet ACTES pour construire un démonstrateur. Notre description intuitive et très générale avait pour but de suggérer que cette approche basée sur les défauts offre un cadre nouveau pour poser les problèmes de vérification interactive de spécifications entre un utilisateur et une base de textes

9. CONCLUSIONS

9.1. Rappel des objectifs

Le projet ACTES avait deux objectifs essentiels : d'une part, la réalisation d'un prototype de poste de travail linguistique orienté vers la compréhension de textes ; d'autre part l'étude du processus d'automatisation de l'extraction de règles à partir de textes de spécification d'un domaine de "l'avionique" écrits en langage naturel.

Le lien entre ces deux objectifs vient du fait que l'extraction de règles était envisagée comme l'aboutissement du processus de compréhension des textes et que cette compréhension automatique devait être réalisée par des modules, traitant les informations linguistiques et pragmatiques, développés à l'aide du poste de travail.

La première phase du projet se "limitait" à la construction de la représentation sémantique intermédiaire de quelques textes représentatifs du domaine à l'aide du prototype de poste de travail et à l'étude, sur papier, du problème de la génération, à partir de cette représentation intermédiaire, des règles correspondantes. Notre thèse couvre cette première phase.

Nous voudrions, sur chacun des deux objectifs, faire la synthèse du travail réellement accompli, de l'intérêt des résultats obtenus, tant du point de vue de la linguistique informatique que de celui de l'expertise des textes de spécification et indiquer quelques axes de travaux futurs. Considérons d'abord le problème du poste de travail.

9.2. A propos du poste de travail

L'intérêt de construire un poste de travail linguistique dans un tel cadre réside dans la mise en oeuvre informatique d'approches générales en linguistique informatique, comme les grammaires d'unification et la Théorie de la Représentation du Discours, à des problèmes de compréhension de textes bien concrets, liés à un corpus défini et cohérent. Il s'agit de savoir si ces approches peuvent être utilisées dans ACTES, quelles sont leurs limitations et de quelles façons elles peuvent être d'intégrées dans un système convivial de façon à prendre en compte les différents niveaux de connaissances présents dans ces textes : syntaxique, sémantique et pragmatique.

Notre poste de travail repose en grande partie sur le formalisme de développement de grammaires, AVAG. Il s'inspire des grammaires d'unification dont l'attrait, d'un point de vue informatique, vient du fait qu'elles se placent à un méta-niveau[12] par rapport à différentes théories du langage naturel. Elles s'attachent à décrire les structures de données et les opérations nécessaires à la mise en oeuvre informatique de grammaires et sont particulièrement sensibles à dégager une sémantique claire pour leurs méta-langages de description de grammaires.

Les grammaires d'unification ne couvrant généralement que les aspects syntaxiques du traitement du langage, nous nous sommes intéressés ensuite à la Théorie de la Représentation du Discours. Cette théorie est séduisante par son adéquation expressive à la représentation de certains problèmes sémantiques dont la portée dépasse le cadre de la simple phrase. Si le noyau initial de cette théorie a le mérite de la simplicité de mise en oeuvre, il laisse cependant de côté des problèmes fondamentaux en compréhension de texte, à savoir la prise en compte du domaine[13]. Le traitement des anaphores du sous-ensemble du langage abordé par la DRT semble également avoir peu de pertinence linguistique.

Une première façon de répondre à ces limitations est d'intégrer partiellement plusieurs théories dans un seul formalisme. Nous avons donc tenté d'inclure dans AVAG certains phénomènes syntaxiques traités dans les grammaires d'unification avec une forme de représentation sémantique intermédiaire empruntée à la DRT et des mécanismes de représentation des connaissances issus du langage hybride BACK. La formulation de tous ces aspects est homogène et la compilation du langage AVAG vise à utiliser au mieux les potentialités de Prolog.

Mais une telle unification a ses limites et AVAG ne pourrait être beaucoup étendu sans devenir rapidement difficile à manipuler informatiquement. Nous avons alors tenté de faire coopérer les différentes théories, dans leurs aspects complémentaires, en les intégrant dans une architecture modulaire. C'est ainsi que nous avons conçu notre module de résolution d'anaphores sous la forme d'un "tableau noir".

Le prototype de poste de travail conçu sur ces principes a été implanté en SP-Prolog sur station graphique UNIX BULL SPS7/300. Il comprend 11000 lignes de code pour les programmes, données (grammaire, lexique, domaine,..) non comprises. Le manuel de l'utilisateur, ainsi que toutes les données sont présentées dans [CHAN 89].

---

[12]Les grammaires d'unification auxquelles nous faisons surtout allusion sont celles que nous avons qualifiées "d'outils linguistiques". Le terme "méta-niveau" ne réfère pas à l'aspect langage de programmation de ces grammaires, mais à la sémantique des structures et opérations utilisées par ces grammaires à partir de laquelle on peut établir des comparaisons entre différentes théories grammaticales (cf [GAZD 88]).

[13] Au sens représentation des connaissances du domaine d'un texte et non au sens logique du terme.

Nous n'avons pas parlé jusqu'à présent d'une des fonctionnalités importantes du poste de travail: l'interface graphique. Elle a sans doute peu de pertinence linguistique, mais sa qualité conditionne en partie la facilité d'utilisation du formalisme, l'affinement des diagnostics d'erreurs[14], la bonne représentation expressive des concepts de l'utilisateur et la souplesse d'enchaînement des différents modules. Programmer dans un environnement multi-fenêtrage fait appel à des techniques appropriées: regroupement des entrées-sorties, suivi des actions de l'utilisateur, contrôle de l'empilement des différents processus appelés,... Ce travail prenant a limité notamment le développement de techniques plus sophistiquées de stratégies d'analyse et d'aides après erreurs dans l'analyseur HANNA.

Quelles sont les réalisations accomplies avec ce poste de travail ? Le formalisme AVAG a servi, dans une de ses premières versions, à réécrire complètement une grammaire en chaîne syntaxique du français, initialement programmée directement en Prolog. Le poste, dans son ensemble, a permis de développer lexique, grammaire, base de connaissances nécessaires au traitement des quelques textes retenus pour le prototype suivant les critères exposés au chapitre 5. Il a servi à les analyser et produire leur représentation sémantique intermédiaire après résolution des principaux cas d'anaphores. Trois personnes ont utilisé le poste de travail de ACTES. Une démonstration des fonctionnalités du système a été faite aux représentants des parties contractantes. Elle a étée jugée très satisfaisante.

Ces quelques éléments permettent-ils de valider le poste de travail ? Pas vraiment car, d'une part, les textes analysés ne sont pas assez nombreux, ni les difficultés assez variées, et, d'autre part, aucune personne extérieure au projet n'a utilisé notre système.

Les critères d'évaluation des systèmes en TLN font d'ailleurs défaut[15]. La plupart des publications dans ce domaine sont d'une grande discrétion sur les résultats obtenus et les moyens de les mesurer. En particulier, les environnements de développement de grammaires n'ont souvent été utilisés que par leurs auteurs ou quelques chercheurs proches. Alors comment mesurer objectivement, par exemple, l'amélioration de rendement dans l'écriture de grammaires que ces environnements sont censés procurer ?

---

[14] une interface n'est pas seulement un procédé technique. Dans le cas d'interface guidée par les menus dans un système de TLN (dont nous avons parlé à propos des diagnostics d'erreur) il peut s'avérer pertinent de relier méta-langage grammatical et langage de description d'interface graphique.

[15] sauf en ce qui concerne les systèmes construits sur des méthodes statistiques de traitement du langage naturel (mais ils sont limités à une "compréhension" superficielle des textes) ou les interfaces en langage naturel pour les base de données (mais il a fallu plus de quinze ans de recherche et les efforts cumulés de nombreuses équipes sur ce point précis).

Les utilisateurs potentiels du poste de travail du système ACTES existent-ils ? Quelles difficultés rencontreraient des linguistes, peu préoccupés par des problèmes d'implantation, pour se servir de notre environnement ? Des experts du domaine arriveraient-ils à représenter une partie de leurs connaissances dans BACK ? Nous manquons d'exprérience pour répondre à ces questions. Que l'on songe simplement que BACK, comme beaucoup d'autres formalismes de représentations des connaissances, n'a sans doute jamais été utilisé par d'autres personnes que des chercheurs en Intelligence Artificielle[16] !

Quels axes de travaux futurs verrions-nous autour du poste de travail linguistique ? Tout au long de l'exposé, nous avons signalé les limites de chacun des traitements décrits et ouvert des perspectives. Voici les points sur lesquels nous voudrions insister (certains ne concernent pas spécialement le projet ACTES, mais présente un intérêt pour la généralisation de solutions particulières avancées dans ce projet):

- ajouts de sources de contraintes supplémentaires dans le module de résolution d'anaphores, en particulier élargissant la gestion du focus global (en liaison avec la structuration du texte). L'intérêt, outre linguistique, serait l'incorporation de techniques de planification, concernant un sous-module, dans l'architecture générale en "tableau noir".
- élaboration d'une véritable grammaire de texte permettant la description de sa structure, extension de la formalisation (au niveau grammatical et RC) des différentes façons de formuler en langage naturel des règles (prise en compte du temps).
- acquisition semi-automatique de connaissances et mots nouveaux: à partir de la reconnaissance par l'analyseur de mots nouveaux (essentiellement des mots composés), analyse de leur contenu, proposition de classification dans le réseau sémantique des concepts correspondants et ajout dans le lexique.

[16] Pour conclure de façon quelque peu provocatrice sur ce point, nous pensons que chercher des éléments de réponses plus précis n'a pas de sens, tant que, d'une part, il ne sera pas admis, hors du milieu de la recherche, que financer un projet de compréhension du LN avec des spécifications limitées présente un intérêt pour le moyen et long terme et que, d'autre part, en retour, des chercheurs n'accepteront pas en plus grand nombre d'appliquer leurs théories sur des corpus réels.

9.3. A propos de l'extraction de règles et en guise de conclusion

Qu'en est-il des réalisations concernant le second objectif du projet ACTES, c'est-à-dire l'extraction de règles à partir des textes de spécifications ? Nous avons, sur ce point, décrit un axe de travail, mais rien implanté. Nous proposons de ne pas générer directement, après l'analyse syntaxico-sémantique d'un texte, les règles de production nécessaires au système expert, mais de passer par une phase intermédiaire de génération de règles de défaut. Cette logique non-monotone modélise bien la façon dont les experts énoncent les règles, par raffinements successifs, éventuellement contradictoires (au sens où une nouvelle phrase, un nouveau paragraphe peuvent exprimer des exceptions par rapport à des parties du texte énoncées antérieurement). Grâce aux méthodes de déduction opérant sur les règles de défaut, il serait possible d'opérer à tout moment de l'énoncé des vérifications de cohérence locale au texte, ce qui est un des objectifs principaux, à long terme, du projet.

Nous avons décrit, sur un exemple simple, comment produire, à partir de la représentation sémantique intermédiaire d'un texte, les règles de défauts et comment traduire ces règles de défaut en règles de production. Il reste un important travail à accomplir (sans doute l'équivalent d'une nouvelle thèse) pour implanter un démonstrateur automatique fonctionnant sur un sous-ensemble de la logique des défauts. Quelques indications sur ce travail ont été données. Cette recherche est relativement indépendante des résultats obtenus dans le traitement du premier objectif[17]. Elle est aussi réaliste, compte-tenu des progrès accomplis dans les recherches sur ce type de logique.

Un tel travail, intégré dans le poste de travail déjà réalisé, apporterait une première contribution concrète au problème de l'extraction de règles à partir de textes en langage naturel, problème dont l'importance a été soulignée par les chercheurs en Intelligence Artificielle préoccupés par les processus d'automatisation de connaissances à partir de textes et par les experts en spécifications, mais dont la résolution en est encore, à l'heure actuelle, à ses premiers balbutiements.

---

[17] Mais l'inverse n'est pas vrai. L'objectif final d'extraction de règles a fortement influé sur le contenu des représentations sémantiques intermédiaires obtenues après analyse des textes, et sur la nature des connaissances à représenter. Il a orienté notre travail dans la mesure où il a fallu déterminer quelles étaient les informations pertinentes dans les textes pour la description de ces règles.

BIBLIOGRAPHIE

ABRÉVIATIONS

Des abréviations sont utilisées pour de nombreux actes de conférences et articles de revues.

AAAI : *American Association for Artificial Intelligence* : Morgan Kaufmann Publishers, Inc., 95 First Street, Los Altos, California, USA, 94022.

IJCAI : *International Joint Conference on Artificial Intelligence* : idem.

ACL : *Annual Meeting of the Association of Computational Linguistics* : Dr. Donald Walker (ACL), Bell Communications Research, 445 South Street, MRE 24379, Morristown, New Jersey, USA, 07960.

COLING : *International Conference on Computational Linguistics* : idem.

CL : *Computational Linguistics* (ancien nom : *American Journal of Computational Linguistics*): idem.

RNLP : *Readings in Natural Language Processing* ; Grosz B., Jones K., Webber B. (ed.), Morgan Kaufmann Publishers, 1986.

TINLAP : *Theoritical Issues in Natural Language Processing*. 3 conférences ont eu lieu sur ce thème dont TINLAP-3 à New Mexico en 1987.

[ACOR 89] Projet ESPRIT ACORD n° P393: *The Construction and Interrogation of Knowledge Base using Natural Language Text and Graphics* ; Des articles seront prochainement publiés dans des revues et conférences. Pour une bibliographie provisoire s'adresser au Centre de Recherche BULL, Louveciennes.

[AHO 86] Aho A., Sethi R., Ullman J.: *Compilers: Principles, Techniques, and Tools* ; Reading, MA , Addison-Wesley, 1986.

[AITK 86] Aït-Kaci H., Nasr R. : "LOGIN: A Logic Programming Language with Build-In Inheritance", *Journal of Logic Programming*, vol.3,n°3, octobre1986.

[AITK 89] Aït-Kaci H., Lincoln P.: "LIFE: A Natural Language for Natural Language", *MCC Technical Report* ; ACA-ST-074-88, février 1988.

[ALLE 87] Allen J.: *Natural Language Understanding* ; Benjamin/Cummings , Californie, 1987.

[ATTA 81] Attardi G, Simi M.: "Consistency and Completeness of Omega, a Logic for Knowledge Representation, *IJCAI 81* ; Vancouver, 1981

[BALZ 85] Balzer R.: " A 15 year perspective on automatic programming"(invited paper) , *IEEE Transactions on Software Engineering* ; SE-11(11), novembre 1985.

[BARH 64] Bar-Hillel Y.: *Language and Information* ; Wesley, Reading Mass., 1964.

[BARW 81] Barwise J., Cooper R.: "Generalized Quantifiers and Natural Language", *Linguistics and Philosophy* ; vol.4, 1981.

[BARW 83] Barwise J., Perry J. : *Situations and Attitudes* ; MIT Press, Cambridge, Massachusetts, 1983.

[BATE 78] Bates M.: "The Theory and Practice of Augmented Transition Networks" in *Natural Language Communication with Computers* ; Bolc L. (ed.), Springer, New York, 1978.

[BATE 87] Bates M., Weischedel R.: *Evaluating Natural Language Interfaces* , Tutorial Session ACL 87; Stanford, CA, juillet 1987.

[BEAU 81] De Beaugrande R., Dressler W.: *Introduction to Text Linguistics* ; Longman Linguistics Library, Londres et New-York, 1981.

[BERI 88] Beringer H.: Traitement automatique des ambiguïtés structurales du Français utilisant la théorie Sens-Texte ; Thèse Université Paris VI, octobre 1988.

[BETK 89] Bethke I., Frey W., Kamp H., Zeevat H.: *Extension of the theorem Prover to Intensional Phenomena* , Deliverable T1.10 ; projet ESPRIT ACORD, n° P393, Université de Stuttgart, mars 1989.

[BIEB 87] Biebow B. & al: "Enrichissement de réseau sémantique à partir de connaissances exprimées en langage naturel", *Congrès AFCET-RFIA* ; Antibes, Novembre 1987.

[BLOC 87] Block R.: *Papers on Reference and Knowledge Representation* ; Projet WISBER, Université de Hambourg, décembre 1987.

[BOGU 87] Boguraev B., Spark-Jones K.: "A note on a study of cases", *CL* ; Vol 13, 1-2, 1987.

[BOGU 88] Boguraev B., Carrol J., Briscoe T., Grover C.: "Software Support for Practical Grammar Development", *COLING 88* ; Budapest, août 1988.

[BRAC 83] Brachman R.J.: "What IS-A Is and Isn't: An Analysis of Taxinomic Links in Semantics Network", *IEEE Computer* ; vol. 16, 10, 1983.

[BRAC 85a] Brachman R.J.,Schmolze J.G.: "An Overview of the KL-ONE Knowledge Representation System", *Cognitive Science*, vol. 9, 1985.

[BRAC 85b] Brachman R.J, Lesveque H.(eds.): *Readings in Knowledge Representation* ; Morgan Kaufman, Los Altos, CA, 1985.

[BRAC 85c] Brachman R.J., Gilbert V.P., Levesque H.J.: "An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of KRYPTON", *IJCAI 85* ; Los Angeles, CA, août 1985.

[BRAC 85d] Brachman R.J.:"I lied about the Trees' or Default and Definitions in Knowledge Representation", *The AI Magazine* ; vol. 6, 3, fin 1985.

[BRAC 88] Brachman R.J., McGuinness D.L.:"Knowledge Representation, Connectionism, and Conceptual Retrieval", *11° Intern. ACM/SIGIR Conference on Research & Development in Information Retrieval* ; Grenoble, juin 1988.

[BRACO 88] Bracon G.: "La spécification de logiciel dans l'industrie" , *Bigre n° 58* ; Actes des journées AFCET sur la Spécification de logiciel, Paris, janvier 1988.

[BRES 82] Kaplan R., Bresnan J. : "Lexical Functional Grammar : A Formal System for Grammatical Representation", in *The Mental Representation of Grammatical Relations* ; Bresnan (ed.), MIT Press, Cambridge, MA, 1982.

[BRIS 87] Briscoe T., Grover C., Boguraev B., Caroll J. : "A Formalism and Environment for the Development of a Large Grammar of English", *IJCAI 87*, Milan , août 1987.

[BURT 79] Burton R.R., Brown J.S.: "Toward a Natural-Language Capability for Computer-Assisted Instruction", in *Procedures for Instructional Systems Development* ; O'Neil H. (ed.), Academic Press, New-York, 1979. Réédité dans RNLP.

[CALD 88a] Calder J., Klein E., Zeevat H.: "Unification Categorial Grammar: A Concise, Extendable Grammar for Natural Language Processing", *ACL 88* ; Budapest, août 1988.

[CALD 88b] Calder J., Klein E., Moens M., Reape M.: "English Parser", *ACORD Deliverable T1.6* ; Projet ESPRIT 393 ACORD "Construction and Interrogation of Knowledge Bases using Natural Language Text and Graphics", février 1988.

[CARB 88] Carbonell J.G., Brown R.D. : "Anaphora Resolution : A Multi-Strategy Approach", *COLING 88* ; Budapest, août 1988.

[CART 87] Carter D. : *Interpreting Anaphors in Natural Language Texts* ; Ellis Horwood, 1987.

[CAST 88] Bonte E., Castaing J., Grandemange P., Grumbach S., Kayser D., Levy F.: "Description générale d'un raisonneur à profondeur variable", Rapport de Recherche LIPN, Université Paris-Nord, 1988.

[CAWL 81] McCawley J.D.: *Everything That Linguits Have Always Wanted to Know about Logic* ; Univ. Chicago Press, Chicago, 1981.

[CHAM 89]. Chambreuil M.: *La Grammaire de Montague: Langage, Traduction, Interprétation* ; ADOSA, Clermont-Fd, juin 1989.

[CHAN 86] Chanier T., Sedogbo C.: *AVAG: Un outil pour les grammaires de traits* ; Rapport du Centre de Recherche BULL, Décembre 1986.

[CHAN 87] Chanier T., Sedogbo C., Salkoff M.: *French Parser*, Deliverable of Task 1.5 ; Projet ESPRIT ACORD, n° P393, février 1987.

[CHAN 88] Chanier T., Fournier C. : "ACTES: Knowledge Acquisition from Texts for a Specification Expert", *Conférence Internationale "Interaction Homme-Machine et Intelligence Artificielle dans les domaines de l'Aéronautique et de l'Espace"* ; Toulouse, septembre 1988.

[CHAN 89] Chanier T., Fournier C. : *Le système ACTES : Formalisme et Environnement , Le prototype, Manuel du poste de travail* ; Rapport du Centre de Recherche BULL, 1989.

[CHAR 73] Charniak E.: "Jack and Janet in Search of a Theory of Knowledge", *IJCAI 73* ; Stanford, CA, 1973.

[CHEN 82] Chenut-Martin S., Doladille F.: "D.L.A.O.: Un système d'aide à la définition de logiciel" , *Actes des journées BIGRE 82* ; Grenoble, janvier 1982.

[CHOM 57] Chomsky N.: *Syntactic Structures* ; Mouton, La Hague, 1957. (*Structures syntaxiques*, Le Seuil, Paris, 1957).

[CHOM 81] Chomsky N.: *Lectures on government and binding* ; Foris, Dordrecht, 1981.

[CHRI 87] Bensadoun O., Chrisment C.: "Huge and Complex Objects: Document Handling", *Recherche et Developpement dans les industries de la langue* ; Cours AFCET, Paris, novembre1987.

[CL 83] *Computational Linguistics* : Special Issue on Ill-Formed Input ; vol. 9, 3-4, juillet-septembre 1983.

[CL 85] *Computational Linguistics*: Special Issue on Machine Translation ; Slocum J. (ed.), vol. 11, 1-3, 1985.

[CL 87] *Computational Linguistics*: Special Issue on the Lexicon ; vol 13, 3-4, 1987.

[COLM 78] Colmerauer Alain : "Metamorphosis Grammars", in *Natural Language communication with computers*, Bolc, Springer-Verlag, Berlin, pp.139-189.

[COLM 79] Colmerauer A.: "Un sous-ensemble interessant du français", *RAIRO Informatique Théorique* ; vol 13, 4, 1979

[COLM 82] Colmerauer A., Kittredge R. : "Présentation du système ORBIS", *COLING 82* ; Prague 1982.

[COLM 83] Colmeraueur A.: "PROLOG, bases théoriques et développements actuels", *Technique et Science Informatique* ; vol. 2, 4, 1983.

[COOP 87] Cooper R.: "Meaning representation in Montague grammar and situation semantics", *Computational Intelligence* ; 3, 1987.

[COUL 86] Coulon D., Kayser D. : "Informatique et langage naturel : Présentation générale des méthodes d'interprétation des textes écrits", *Technique et Science Informatique*, vol 5, n° 2, 1986.

[CULL 81] Cullingford R.: "SAM" in *Inside Computer Understanding* ; Schank R. , Riesbeck C. (ed), Hillsdale, N.J., Lawrence Erlbaum, 1981, pp 75-89.

[DAHL 86] Dahl D.A.: "Focusing and Reference Resolution in PUNDIT", *AAAI 86* ; Philadelphie, PA, août1986.

[DANL 85] Danlos L.: *Génération automatique de textes en langues naturelles* ; Masson, 1985.

[DEJO 82] Dejong G.: "An Overview of the FRUMP System", in *Strategies for Natural Language Processing*, ; Lehnert W.G., Ringle M.H. (ed.), Lawrence Erlbaum Associates, 1982.

[DEKL 86] De Kleer J.: "An Assumption-Based TMS", Artificial Intelligence ; 28, 1986.

[DINC 88] Dincbas M., Van Hentenryck P., Simonis H., Aggoun A., Graf T., Berthier F. : "The Constraint Logic Programming Language Chip", *International Conference on Fifth Generation Computer Systems*; ICOT (ed.), Tokyo, décembre 1988.

[DOYL 79] Doyle J. : "A Truth Maintenance System" ; *Artificial Intelligence*, 12, 1979.

[EIJC 84] van Eijck J.: "Discourse Representation Theory and Plurality" in *Studies in model theoretic semantics* ; Meulen A. (ed.), Groningen, Amsterdam, 1983

[EISE 86] Eisele A., Dörre J. : "A Lexical Functional Grammar System in Prolog", *COLING 86 ;* Bonn, 1986.

[EISE 88] Eisele A., Dörre J. : "Unification of Disjonctive Feature Descriptions Structures", *ACL 88*; Buffalo, NY, juin 1988.

[ENGE 88] Engelmore R.S., Morgan A.J.: *Blackboard Systems* ; Addison-Wesley, 1988.

[ETHE 87] Etherington D. W.: "Formalizing Nonmonotonic Reasoning" ; *Artificial Intelligence*, 31, 1987.

[EVAN 85] Evans R.: "ProGram : a development tool for GPSG grammars", *Linguistics* ; vol. 23, 2, 1985.

[FARG 86] Fargues J., Landau M.C., Duguourd A., Catach L.: "Conceptual graphs for semantics and knowledge processing", *IBM Journal of Research and Development* ; vol. 30, 1, janvier 1986.

[FAUC 84] Fauconnier Gilles: *Espaces mentaux* ; Editions de Minuit, Paris 1984.

[FILL 68] Fillmore C.:"The case for case" in *Universals in Linguistic Theory* ; Bach E., Harms R. (eds.), New-York:Holt,Rinehart,and Winston, 1968.

[FILM 88] Filman R.E. : "Reasoning with Worlds and Truth Maintenance in a Knowledge-based Programming Environment", *Communications of the ACM*, Vol 31, 4, Avril 1988.

[FOUQ 88] Fouqueré C.: *Systèmes d'analyse tolérante du langage naturel* ; Thèse Université Paris-Nord, janvier 1988.

[FROI 88] Froidevaux C, .Kayser D.: "Inheritance in Semantics Networks and in Default Logic", *Non Standard Logics for Automated Reasoning*, Smets P.& al, Academic Press, 1988.

[GAZD 85] Gazdar G, Klein E., Pullum G., Sag I. : *Generalized Phrase Structure Grammar* ; London : Blacwell, 1985.

[GAZD 87] Gazdar G.: "The new Grammar Formalisms : A tutorial Survey", *IJCAI 87* ; Milan, aôut 1987.

[GAZD 87b] Gazdar G.: "COMIT => PATR-II" *TINLAP-3: Theoretical Issues in Natural Language Processing* ; New Mexico, janvier 1987.

[GAZD 88] Gazdar G., Pullum G.K., Carpenter R., Klein E., Hakari T.E., Levine R.D. : "Category structures", *CL* ; Vol 14, 1, fin 1988.

[GIAN 85] Giannesini F., Kanoui H., Pasero R., Van Caneghem M.: *Prolog* ; InterEditions, 1985.

[GRAN 87] Granacki J.J., Parker A.C., Arens Y.:"Understanding system specifications written in natural language" , *International Joint Conference on Artificial Intelligenc*e *IJCAI* ; Milan, août 1987.

[GRIS 86] Grishman R., Kittredge R. (eds): *Analyzing Language in Restricted Domains: Sublanguage Description and Processing* ; Lawrence Erlbaum, Hillsdale,NJ, 1986.

[GROS 75] Gross M.: *Méthodes en Syntaxe* ; Hermann, Paris, 1975.

[GROS 84] Gross M. : "Lexicon-Grammar and the Syntactic Analysis of French" *COLING et ACL 84* ; Stanford, CA, juillet 1984.

[GROS 86] Gross M. :"The representation of Compound Words", *COLING 86* ; Bonn, août 1986.

[GROSZ 77] Grosz B.: "The Representation and Use of Focus in a System for Understanding Dialogs", *IJCAI 77* ; Cambridge, MA, 1977. Réédité dans RNLP.

[GROSZ 83] Grosz B., Joshi A.K., Weinstein S. : "Providing a Unified Account of Definite Noun Phrase in Discourse", *ACL 83*, 1983.

[GROSZ 86] Grosz B., Sidner C. : "Attention, Intentions, and the Structure of Discourse", *AJCL* ; Vol 12, 3, juil-sept 1986.

[GUEN 83] Guenthner F., Lehman H. : "Rules for Pronominalization", *European Chapter of the ACL 83* ; Pise, 1983.

[GUEN 86] Guenthner F., Lehman H., Schonfeld W. : "A Theory for the Representation of Knowledge", *IBM Journal of Research and Development*; 30(1), janvier 1986.

[GUNJ 87] Gunji T.: *Japanese Phrase Structure Grammar* ; Dordrecht D. Reidel, 1987.

[HAHN 87] Hahn U.: "Modeling Text Understanding - The Methodological Aspects of Automatic Acquisition of Knowledge through Text Analysis"(invited paper), *1st International Symposium on Artificial Intelligence and Expert Systems* ; Berlin, mai 1987.

[HALL 85] Halliday M., Fawcett R.: *New Developments in Systemic Linguistics*; Batsford, Londres, 1985.

[HARR 51] Harris Z.: *Methods in Structural Linguistics* ; Univ. Chicago Press, Chicago, 1951.

[HARR 68] Harris Z.:*Mathematical Structure of Language* ; New York ,Wiley, 1968. (Structures mathématiques du langage ; Dunod, Paris,1971).

[HEIM 82] Heim I.: *The Semantics of Definite and Indefinite Noun Phrases*. Ph.D. dissertation, Univ. of Massachussets, Amherst, 1982.

[HEND 78] Hendrix G.: "Semantic Aspects of Translation", in *Understanding Spoken Language* ; Walker D. (ed.), Elsevier Science Publishing Company, New-York, 1978. Réédité dans RNLP.

[HEYE 88] Heyer G., van Hoof T., Luther K., Ovenhausen P.: *Collaboration between Domain and Dialog Manager*, Deliverable T5.10 ; projet ESPRIT ACORD, n° P393, Triumph-Adler, 1988.

[HIRS 88] Lang F.M., Hirschman L.: "Improved Portability and Parsing through INteractive Acquisition of Semantic Information", *2° Conference on Applied Natural Language Processing* ; Austin, février 1988.

[HIRST 81] Hirst G. : *Anaphora in Natural Language Understanding*, Springer-Verlag, 1981.

[HOBB 78] Hobbs J. : "Resolving Pronoun References", *Lingua* 44, 1978 ; North-Holland. Réédité dans RNLP.

[ICOT 86] "Research Activities on Natural Language Processing of the FGCS Project", *Fifth Generation Computer Systems ICOT Journal* ; N° 14, juin 1986.

[ICOT 87] "Research Activities at ICOT Second Research Laboratory", *Fifth Generation Computer Systems ICOT Journal* ; N° 16, juin 1987.

[IORD 88] Iordanskaja L., Kittredge R., Polguère A.: "Implementing a Meaning-Text Model for Language Generation", *supplément aux actes du congrès COLING 88* ; Budapest, août 1988.

[JACO 78] Jacobson B.: , *Transformational-generative Grammar* ; North-Holland, Amsterdam, 1978.

[JAFF 84] Jaffar J.: "Efficient unification over infinite terms", *New Generation Computing* ; vol. 2, 3, 1984.

[JAFF 87] Jaffar J., Lassez J.: "Constraint Logic Programming", *ACM Symposium on Principles of Programming Languages* ; Association for Computing Machinery, Munich, 1987.

[JOSH 85] Joshi A. : "Tree-adjoining grammars: How much context sensitivity is required to provide reasonable structural descriptions", in *Natural Language Parsing* ; Dowty D.R., Karttunen L., Zwicky A. (eds.), Cambridge Univ. Press, New-York, 1985.

[KAMP 81] Kamp H. : "A Theory of Truth and Semantic Interpretation", in *Formal Methods in the Study of Language*, Groenendijck (ed.), Amsterdam, 1981.

[KAMP 88] Kamp H.: "Discourse Representation Theory: What it is and where it ought to go", in *Natural Language at the Computer* ; Blaser A. (ed.), Lecture Notes in Computer Science, Springer-Verlag, 1988.

[KAPL 73] Kaplan R.M.: "A general syntactic processor" in *Natural Language Processing* ; Rustin R. (ed.), Algorihmics Press, New York, 1973.

[KART 76] Karttunen L. : "Discourse Referents", in *Syntax and Semantics* ; Vol. 7 , McCawley J. (ed.), Academic Press, 1976.

[KART 84] Karttunen L. : "Features and Values", *COLING 84* ; Stanford, CA, juillet 1984.

[KART 84] Shieber S.M., Karttunen L., Pereira F. : *Notes from the Unification Underground : A compilation of Papers on Unification-based Grammar Formalisms* ; Technical Report 327, SRI International , Californie, 1984.

[KART 86] Karttunen L. : "D-PATR : A Development Environment for Unification-Based Grammars", *COLING 86 ;* Bonn, août1986

[KASP 86] Kasper R.T., Rounds W.C. : "A Logical Semantics for Feature Structures", *ACL 86* , New-York, NY, juin 1986.

[KASP 87 a] Kasper R.T.: *Feature Structures : A Logical Theory with Application to Language Analysis* ; PHD dissertation, University of Michigan, 1987.

[KASP 87 b] Kasper R.T.: "A Unification Method for Disjonctive Feature Descriptions", *ACL 87* , Stanford, CA, juillet 1987.

[KASP 88] Kasper R.T. : "Conditional Descriptions in Functional Unification Grammar", *ACL 88* ; Buffalo, NY, juin 1988

[KAY 79] Kay M.,: "Functional Grammar", *Proc. 5th Annual Meeting of the Berkeley Linguistics Society* ; Berkeley, CA, février 1979.

[KAY 82] Kay M. : "Parsing in Functional Unification Grammar", in *Natural Language Parsing* ; Dowty D., Karttunen L., Zwicky A. (ed.), Cambridge University Press, Angleterre, 1982.

[KAYS 87] Kayser D.: "Une sémantique qui n'a pas de sens", *Langages*, numéro spécial sur *Sémantique et Intelligence Artificielle* ; septembre 1987.

[KEOW 87] McKeown K.R., Paris C.: "Functional Unification Grammar Revisited", *ACL 87* ; Stanford, CA, juillet 1987.

[KITT 82] Kittredge R., Lehrberger J. (eds): *Sublanguage: Studies of Language in Restricted Domains* ; De Gruyter, New-York, 1982.

[KITT 86] Grishman R., Kittredge R., editors: *Analysing Language in Restricted Domains: Sublanguage Description and Processing*. Lawrence Erlbaum, Hillsdale, NJ, 1986.

[KLEI 84] Klein E. : "VP Ellipsis in DR Theory" in *Studies in Discourse Representation Theory and the Theory of Generalized Quantifiers* ; Groenendijk, de Jongh, Stokhof (ed.), Foris Publication, Groningen, Amsterdam, pp 161-187, 1984.

[KLEI 86] Klein E., Johnson M. : "Discourse, anaphora and parsing", *COLING 86 ;* Bonn, août1986.

[ISAB 84] Isabelle P.: "An Other Look at Nominal Compounds", *COLING 84* ; Stanford, 1984.

[LANC 88] Lancel J.M., Otani M., Simonin N., Danlos L.: "Analyse et génération de phrases dans le cadre d'un dialogue en langage naturel", *Colloque Informatique et Langue Naturelle* ; Nantes, octobre 1988.

[LEHM 88] Lehman H.: "The LEX-Project - Concepts and Results", in *Natural Language at the Computer* ; Blaser A. (ed.), Lecture Notes in Computer Science, Springer-Verlag, 1988.

[LESV 87] Levesque H.J., Brachman R.J.: "Expressiveness and Tractability in Knowledge Representation and Reasoning", *Computational Intelligence* ; vol.3, 2, printemps 1987.

[LUCK 86] Luck K., Nebel B., Petalson C., Schmiedel A.: *The Anatomy of the BACK-System* ; Rapport Interne Technische Universität, Berlin ,1986.

[LYTI 86] Lytinen S., Gershman A.: "ATRANS: Automatic Processing of Money Transfer Messages", *AAAI 86* ; Philadelphie, 1986.

[MARC 78] Marcus M.: "A Computational Account of Some Constraints on Language", *Theoretical Issues in Natural Language Processus, TINLAP 2* ; Waltz D. (ed.), édité par ACL, Urbana-Champaign 1978. Réédité dans RNLP.

[MCOR 87] McCord M.: "Natural Language Processing in PROLOG" in *Knowledge Systems and Prolog* ; Walker A., McCord M., Sowa J.F., Wilson W.G. (eds.), Addison-Wesley, 1987.

[MAEDA 88] Maeda H. & al : "Parsing Japanese Honorifics in Unification Based Grammar", *ACL 88* ; Buffalo, NY, juin 1988.

[MELC 70] Mel'cuk I.A., Zolkovskij A. K.: "Towards a Functioning 'Meaning-Text' Model of Language", *Linguistics* ; n° 57, 1970.

[MELC 87] Mel'cuk I.A., Polguère A.: "A Formal Lexicon in the Meaning-Text Theory", *CL* ; vol. 13, 3-4, Juillet-décembre 1987.

[MELL 88] Mellish C.S. : "Implementing Systemic Classification by Unification", *CL* ; Vol 14, 1, fin 1988.

[MONT 74] Montague R. : "The proper Treatment of quantification in ordinary English", in *Formal Philosophy* ; Thomason R.H. (ed.), Yale University Press, 1974.

[MORI 85] Morin J.Y. : *Theories syntaxique et théorie du parsage* ; Thèse Université Aix-Marseille, Faculté des Sciences de Luminy, 1985.

[MOSH 87] Moshier M.D., Rounds W.C.: "A Logic for Partially Specified Data Structures", *ACM Conference on Principies of Programming Languages* ; Munich, 1987.

[OMMA 88] Ommani F., Sedogbo C.: "Principles and Implementation of a Theorem Prover Based on the Tableau Calculus", *Journées européennes sur les méthodes logiques en IA, JELIA 88* ; Roscoff, juin 1988.

[PALM 86] Palmer M.S., Dahl D.A., Shiffman R.J., Hirschman L., Linebarger M., Dowding J.: "Recovering Implicit Information", *ACL 86* ; New-York, juillet 1986.

[PATE 89] Patel-Schneider P.F.: "A Four-Valued Semantics for Terminological Logics", *Artificial Intelligence* ; vol. 38, 3, avril 1989.

[PELT 87] Peltason C., Luck K., Nebel B., Schmiedel A.: *The User's Guide to the BACK System* ; Rapport Interne Technische Universität, Berlin , janvier 1987.

[PERE 80] Pereira F.C.N. , Warren D. : "Definite Clause Grammar for Language Analysis : A survey of the formalism and a comparison with Augmented Transition Networks", *Artificial Intelligence*, 13, 3., 1980.

[PERE 82] Pereira F., Warren D.: "An Efficient and Easily Adaptable System for Interpreting Natural Language Queries", *JCL* ; vol.8, 3-4, 1982.

[PERE 84] Pereira F.C.N., Shieber S.M.: "The Semantics of Grammar Formalisms Seen as Computer Languages", *ACL et COLING 84* ; Stanford, juillet 1984.

[PERE 87a] Pereira F.C.N. : "Grammars and Logics of Partial Information", *Proc. of the International Conference on Logic Programming* ; Melbourne, mai 1987.

[PERE 87b] Pereira F.C.N., Shieber S.M.: *Prolog and Natural Language Analysis* ; CSLI Lecture Notes ; Center for the Study of Language and Information, Stanford University, 1987. Distribué par University of Chicago Press.

[PERE 88] Pereira F.C.N., Pollack M.E. : "An Integrated Framework for Semantic and Pragmatic Interpretation", *ACL 88* ; Buffalo, NY, juin 1988.

[PIER 87] Pierrel J.M.: *Dialogue oral homme-machine* ; Hermes, Paris, 1987.

[POLL 87] Pollard C., Sag I. : *Information-based Syntax and Semantics* ; CSLI Lecture Notes ; Center for the Study of Language and Information, Stanford University, 1985. Distribué par University of Chicago Press.

[PORT 87] Porter H.H. : "Incorporating Inheritance and Features Structures into a Logic Grammar Formalism", *ACL 87* , Stanford, CA, juillet 1987.

[PUJI 88] Pujin J.M.: *Contribution à l'étude des Raisonnements Non-Monotones: Le Tableur Logique* ; Thèse université Paris VII, juin 1988.

[RAU 88] Jacobs P.S., Rau L.F.: "Natural Language Techniques for Intelligent Information Retrieval", *11° ACM/SIGIR Conference on Research & Development in Information Retrieval*; Grenoble, juin 1988.

[REIN. 83] Reinhart T.: *Anaphora and Semantic Interpretation* ; University of Chicago Press, 1983.

[REIT 80] Reiter R.: "A Logic for Default Reasoning" ; *Artificial Intelligence*, 13,(1,2), 1980.

[RIAO 88] *Conference on User-Oriented, Content-Base Text and Image Handling*, MIT, Cambridge, mars 1988.

[RICH 88] Rich E., Luperfoy S.: "An Architecture for Anaphora Resolution", *2° ConF. on Applied Natural Language Processing* ; Austin, février 1988.

[RNLP 86] *Readings in Natural Language Processing* ; Grosz B., Jones K., Webber B. (ed.), Morgan Kaufmann Publishers, 1986.

[RNNR 87] *Readings in Nonmonotonic Reasoning* ; Ginsberg M.L. (ed.), Morgan Kaufmann, 1987.

[ROBI 82] Robinson J.: "DIAGRAM: A Grammar for Dialogues", *Communication of the Association for Computing Machinery CACM*; vol. 25, 1, 1982. Réédité dans RNLP.

[ROUN 86] Rounds W.C., Kasper R.: "A Complete Logical Calculus for Record Structures Representing Linguistic Information", *IEEE Symposium on Logic in Computer Science* ; Cambridge, MA, juin 1986.

[ROUS 84] Rousselot F.: *Réalisation d'un programme comprenant des textes en utilisant un formalisme unique pour représenter toutes les connaissances nécessaires* ; Thèse d'état, Université Pierre et Marie Curie, 1984.

[SABA 88] Sabah G. : *L'intelligence Artificielle et le langage : Volume 1, représentation des connaissances*; Hermes 1988.

[SABA 89] Sabah G. : *L'intelligence Artificielle et le langage : Volume 2, Processus de compréhension* Hermes 1989.

[SABAT 87] Sabatier P.: *Contribution au développement d'interfaces en langage naturel* , Thèse d'Etat Université Paris 7 ; juillet 1987.

[SAEKI 87] Saeki M., Horai H., Toyama K., Uematsu N., Enomoto H.: "Specification Framework Based on Natural Language", *Fourth International Workshop on Software Specification and Design* ; Monterey, CA, avril 1987.

[SAGE 81] Sager N.: *Natural Language Information Processing: A Computer Grammar of English and its Application* ; Addisson-Wesley, Reading,Mass., 1981.

[SAIN 86] Saint-Dizier P.: "An Approach to Natural-Language Semantics in Logic Programming", *Journal of Logic Programming* ; vol. 4, 1986.

[SAIN 88] Saint-Dizier P.:"Default Logic, Natural Language and Generalized Quantifiers", *COLING 88* ; Budapest, août 1988.

[SALK 73] Salkoff M: *Une grammaire en chaîne du français. Analyse distributionnelle* ; Dunod, Paris, 1973.

[SALT 89] Salton G.: *Automatic Text Processing* ; Addison Wesley, 1989.

[SCHA 77] Schank R., Abelson R.: *Scripts, plans, goals and understanding* ; Hillsdale, N.J., Lawrence Erlbaum, 1977.

[SCHA 80] Schank R. : "Langage and Memory", *Coginitive Science* ; 4(3),1980, pp 242-284. Réédité dans RNLP.

[SCHA 87] Schank R., Kass A.: "Natural Lanfguage Processing: What is Really Involved ? " *TINLAP-3: Theoretical Issues in Natural Language Processing* ; New Mexico, janvier 1987.

[SCHM 86] Schmiedel A., Peltason C., Nebel B., Luck K.v.: *A case Study in Knowledge Representation* ; KIT-Report N° 39, rapport interne Technische Universität, Berlin, 1986.

[SCHU 84] Schubert L., Pelletier F.J.: "From English to Logic: Context-Free Computation of 'Conventional Logical Translations", *JCL* ; vol. 10, 1984. Réédité dans RNLP.

[SEDO 85] Sedogbo C. : "A meta-grammar for handling coordination in Logic Grammars", in *Natural Language Understanding and Logic Programming* ; Dahl V., St Dizier (ed.), North-Holland, 1985.

[SEDO 86] Sedogbo C.: "Extending the Expressive Capacity of the Semantic Component of the OPERA System", *COLING 86 ;* Bonn, août1986.

[SEDO 87] Sedogbo C.: *De la grammaire en chaîne du français à un système de question-réponse* ; Thèse d'Etat, Université d'Aix-Marseille-II, juin 1987.

[SELI 85] Seligmann L., Taillibert P., Vivancos A.: "Définition de logiciel en langue naturel", *Journée de Synthèse Langage Naturel, Congrès AFCET, RFIA* ; Grenoble, novembre 1985.

[SELL 85a] Sells P. : *Restrictive and Non-Restrictive Modification*, CSLI Report, 1985.

[SELL 85b] Sells P. : *Lectures on Contemporary Syntactic Theories* ; CSLI Lecture Notes ; Center for the Study of Language and Information, Stanford University, 1985. Distribué par University of Chicago Press.

[SELM 89] Selman B.: "Connectionist systems for natural language understanding", *Artificial Intelligence Review* ; vol. 3, 1, 1989.

[SHAP 86] Sterling L., Shapiro E.: *The Art of PROLOG* ; Shapiro (ed.), MIT Press, 1986.

[SHIE 83] Shieber S.M., Uskoreit H., Pereira F., Robinson J., Tyson : "The formalism and implementation of PATR-II", in *Research on Interactive Acquisition and Use of Knowledge* ; Grosz B., Stieckel M. (ed.), 1983.

[SHIE 84] Shieber S.M. : "The Design of a Computer Language for Linguistics Information", *COLING 84* ; Stanford, CA, juillet 1984.

[SHIE 86] Shieber S.M. : *An introduction to unification-based approaches to grammar*, CSLI Lecture Notes ; Center for the Study of Language and Information, Stanford University, 1986. Distribué par University of Chicago Press.

[SHIE 87] Shieber S.M. : "Separating Linguistic Analysis from Linguistic Theories" in *Linguistic Theory and Computer Applications* ; Whitelock, Peter J. (eds.) , Academic Press, London, 1987.

[SIDN 79] Sidner C. L.: *Towards a Computational Theory of Definite Anaphora Comprehension in English Discourse*, MIT-AI TR-537, Cambridge, MA, 1979.

[SIDN 83] Sidner C.: "Focusing in the Comprehension of Definite Anaphora", in *Computational Models of Discourse* ; Brady M., Berwick R. (ed.), MIT Press, Cambridge, MA, 1983. Réédité dans RNLP.

[SIMO 85] Simonin N.: *Utilisation d'une expertise pour engendrer des textes structurés en français* ; Thèse d'université, Université Pierre et Marie Curie, 1985.

[SMIT 82] Smith B.C.: *Reflection and Semantics in a Procedural Language* ; Thèse et rapport technique MIT/LCS/TR-272, MIT, Cambridge, MA, 1982.

[SMOL 88] Smolka G. : *A Feature Logic with Subsorts* ; LILOG-Report, IBM-Deutschland, Stuttgart, mai 1988.

[SMOL 89] Smolka G., Aït-Kaci H.: "Inheritance Hierarchies: Semantics and Unification", *Journal of Symbolic Computation* ; Special Issue on Unification, Kirchner C. (ed.), mars 1989.

[SUGI 88] Sugimura R., Hasida K., Akasaka K., Hatano K., Kubo Y., Okunishi T.: "A Software Environment for Research into Discourse Understanding Systems", *International Conference on Fifth Generation Computer Systems* ; Tokyo, décembre 1988.

[TENA 83] Tenant H.R. & al.: "Menu-based Natural Language Understanding", *ACL 83* ; Cambridge, MA, 1983.

[USZK 86] Uszkoreit H. : "Categorial Unification Grammars", *COLING 86 ;* Bonn, août1986.

[USZK 88] Bouma G., König E., Uszkoreit H. : "A flexible graph-unification formalism and its application to natural-language processing", *IBM Journal of Research and Development*; 32, n°2, mars 1988.

[VAUQ 85] Vauquois B., Boitet C.: "Automated Translation at Grenoble University", *CL* ; vol. 11, 1, 1985.

[VILA 85] Vilain M.: "The restricted Language Architecture of a Hybrid Representation System", *IJCAI 85* ; Los Angeles, CA, août 1985.

[VIRB 82] Virbel J.: "La composante matérielle des structures textuelles", LSI-CNRS Toulouse ?

[WADA 86] Wada H., Asher N.: "BUILDRS : An Implementation of DR Theory and LFG", *COLING 86 ;* Bonn, août1986

[WADA 87] Wada H., Asher N. : "A Computational Account of Syntactic, Semantic and Discourse Principles for Anaphora Resolution", *Non-Publié (?)* ; University of Texas at Austin, TX, janvier 1987.

[WALT 87] Waltz D.L.: "Connectionist Models : Not just a Notational Variant, Not a Panacea", *TINLAP-3: Theoretical Issues in Natural Language Processing* ; New Mexico, janvier 1987.

[WEBB 83] Webber B.: "So what Can We Talk About Now ?", in *Computational Models of Discourse* ; Brady M., Berwick R. (ed.), MIT Press, Cambridge, MA, 1983.

[WEBB 88] Webber B.: "Discourse Deixis: Reference to Discourse Segments", *ACL 88*, Buffalo, NY, juin 1988.

[WILK 75] Wilks Y.A.: "An Intelligent Analyser and Understander of English", *Communications of the A.C.M.* ; vol. 18, 1975. Réédité dans RNLP.

[WILK 78] Wilks Y.A.: "Making Preferences More Active", *Artificial Intelligence* ; vol. 11, 1978.

[WILK 85] Wilks Y.A., Huang X., Fass D.C.: "Syntax, Preference and Right Attachment", *IJCAI 85* ; Los Angeles, CA, 1985.

[WINO 73] Winograd T.: "A Procedural Model of Language Understanding" in *Computer Model of Thought and Language* ; Schank R., Colby (eds.), W.H. Freeman, New-York, 1973. Réédité dans RNLP.

[WINO 83] Winograd T.: *Language as a Cognitive Process* ; Addison-Wesley, 1983.

[WOOD 78] Woods W. A.: "Semantics and Quantification in Natural Language Question Answering", *Advances in Computers* ; vol. 17, Yovits M. (ed.), Academic Press, 1978. Réédité dans RNLP.

[WOOD 80] Woods W.A.: "Cascaded ATN grammars", *CL*; 6,1, 1980.

[ZOCK 88] Zock M., Sabah G. (eds.): *Advances in Natural Language Generation, An Interdisciplinary Perspective* ; Communication in AI Series, Pinter Publishers, Londres, vol. 1 et 2, 1988.

ANNEXES

I. EXTRAIT DE LA DRS DU TEXTE PROTOTYPE AVANT LA RÉSOLUTION DES ANAPHORES

Voici une partie de la représentation sémantique (sous forme de DRS) du texte de la figure 8.4, produite par l'analyseur syntaxico-sémantique HANNA.

Le texte correspond à un chapitre (*chap^1*) du pavé Train et Trappes (*paveTT*). Il est composé de cinq parties, successivement:

- titre -> *part^1*
- Règle principale -> *part^2*
- Contrôles sur alarme -> *part^3*
- Forçage -> *part^4*
- Effets de bord -> *part^5*

L'analyseur produit une autre structure mettant en correspondance les phrases et les numéros des DRS associées, ainsi que les arbres syntaxiques. La liste des couples d'index de phrases (le titre étant considéré comme une phrase) et des DRS correspondantes est la suivante:

(ph^1,drs^2) , (ph^2,drs^4) , (ph^3,drs^7) , (ph^4,drs^11) , (ph^5,drs^14) , (ph^6,ds^17) , (ph^7,drs^20) , (ph^8,drs^23) , (ph^9,drs^27) , (ph^10,drs31)

Les parties *univers* et *condition* des DRS sont séparées par un trait. Les marqueurs variables des univers sont des anaphores et vont donner lieu à une recherche d'antécédents. Les DRS modifiées par le module de résolution de anaphores sont présentées dans la deuxième partie.

Rappelons que le nom des marqueurs dans les DRS sont les noms des concepts BACK du domaine. Ainsi *dgear_nd* est le nom du concept représentant l'information *Disc-Gear-Not-Down*.

```
paveTT:
        [chap^1]
chap^1:
        [part^1,part^2,part^3,part^4,part^5]
part^1:
        [drs^1,drs^2]
```

La première DRS de chaque partie est la DRS principale de la partie. Elle n'est composée que d'une partie ***univers****. Ses marqueurs sont accessibles à toutes les sous-DRS de la partie.*

```
drs^1:
        phrase(1)
          dgear_nd
          algear_nd
          [et(algear_nd,dgear_nd)-sortie^X4795/X4792,fem,plu,X4800,X4802,X4804,X4806]
        -------------
```

Une conjonction de coordination entre deux groupes nominaux introduit un marqueur individuel pour chacun des groupes et un marqueur de type ensemble.

drs^2:
[et(algear_nd,dgear_nd)-sortie^X5194/X5191,fem,plu,X5199,X5201,X5203,X5205]
algear_nd
dgear_nd
---
sortie(et(algear_nd,dgear_nd))
part^2:
[drs^3,drs^4,drs^7]
drs^3:
phrase(2)
dgear_nd
algear_nd
hrs
ang_m_dr
ang_m_ga
vit_conv
tdr_ver_b
tga_ver_b
tav_ver_b
cas^1
---
drs^4:
[[cas^1],imp(drs^5,drs^6)]
drs^5:
et(condition^7,et(condition^6,et(condition^5,et(condition^4,et(condition^3,et(condition^2,et(condition^1,X6608)))))))-cas^1/X6614
tav_ver_b
tga_ver_b
tdr_ver_b
vit_conv
ang_m_ga
ang_m_dr
hrs
id6-event^6/X6639
---
eg(tav_ver_b, not ver_b)-condition^7/X6657
eg(tga_ver_b, not ver_b)-condition^6/X6673
eg(tdr_ver_b, not ver_b)-condition^5/X6689
inf(vit_conv,vmini)-condition^4/X6703
inf(ang_m_ga,ang_m_mini)-condition^3/X6717
inf(ang_m_dr,ang_m_mini)-condition^2/X6731
inf(hrs,hrs_mini)-condition^1/X6745
avoir(id6,et(condition^7,et(condition^6,et(condition^5,et(condition^4,et(condition^3,et(condition^2,et(condition^1,X6608)))))))
simultanement(id6,X6800)
ref_dis(id6,X6800)
drs^6:
[et(algear_nd,dgear_nd)-X8540^X8541/X8538,fem,plu,X8546,X8548,X8550,suj]
algear_nd
dgear_nd
id5-event^5/X8565
---
positionner(id5,et(algear_nd,dgear_nd),alarme)
drs^7:
[[],imp(drs^8,drs^9)]
drs^8:
X9306-cas^X9313/X9310
---
not X9306

drs^9:
[X9629-sortie^15/X9633,fem,plu,def,X9643,X9645,suj]
id10-event^10/X9656
---
sortie(X9629)
positionner(id10,X9629, not alarme)
part^3:
[drs^10,drs^11,drs^14,drs^17,drs^20,drs^23]
drs^10:
phrase(4)
vit_conv
vit_sol
cas^2
phrase(5)
vit_sol
cas^3
phrase(6)
hrs
al_b_inert
cas^4
phrase(7)
al_b_inert
cas^5
phrase(8)
cas^6
---
drs^11:
[[cas^2],imp(drs^12,drs^13)]
drs^12:
condition^8-cas^2/X11595
[vit_conv-X11610^X11611/X11608,fem,sing,X11616,X11618,X11620,suj]
id13-event^13/X11631
---
etre(id13,vit_conv,invalide)-condition^8/X11648
drs^13:
id14-event^14/X12319
[vit_sol-X12334^X12335/X12332,fem,sing,X12340,X12342,X12344,obj]
---
utiliser(id14,vit_sol,X12355
drs^14:
[[cas^3],imp(drs^15,drs^16)]

*L'adverbe "également" est traduit par une relation entre deux parties du discours de type **condition**. La première condition est dans la phrase de l'adverbe. La seconde est introduite sous forme de référent à résoudre dans la partie **univers**. Cette anaphore est de type **local**, c'est-à-dire que son antécédent est à rechercher dans la même phrase ou dans la phrase précédente, indépendemment des conditions d'accessibilité logique. De plus les deux marqueurs de type **condition** sont marqués comme disjoints. L'adverbe "simultanément" est traité d'une façon identique et relie deux marqeurs de type **événement**.*

drs^15:
condition^10-cas^3/X13094
[vit_sol-X13109^X13110/X13107,fem,sing,X13115,X13117,X13119,suj]
id17-event^17/X13130
[X13139-condition^X13146/X13143,fem,sing,local,X13153,av,X13157]
---
etre(id17,vit_sol,invalide)-condition^10/X13172
egalement(condition^10,X13139)
ref_dis(condition^10,X13139)

On considère que "c'est-à-dire" établit une référence entre deux conditions.

```
drs^16:
        id18-event^18/X14061
        [X14070-condition^14/X14074,fem,sing,def,X14084,X14086,suj]
        [X14095-vitesse^9/X14099,fem,sing,def,X14109,X14111,obj]
        -------------
         inf(X14095,vmini)-condition^15/X14127
         co_ref(X14070,condition^15)
         conside2rer(id18,X14070,rempli)
         vitesse(X14095)
         condition(X14070,X14095)
part^5:
        [drs^30,drs^31]
drs^30:
        phrase(10)
         tnon_sort
         id38-event^38/X22731
         X22738-event^X22745/X22742
         [X22751-valeur^3/X22755,fem,sing,def,X22765,X22767,obj]
         [X22776-sortie^X22783/X22780,X22784,plu,def,2,X22792,obj]
        ------------
drs^31:
        id38-event^38/X23510
        X23517-event^X23524/X23521
        [tnon_sort-info^X23537/X23534,fem,sing,X23542,X23544,X23546,obj]
        [X23555-valeur^3/X23559,fem,sing,def,X23569,X23571,obj]
        [X23580-sortie^X23587/X23584,X23588,plu,def,2,X23596,obj]
        -------------
         positionner(id38,tnon_sort,X23555)
         simultanement(id38,X23517)
         ref_dis(id38,X23517)
         info(tnon_sort)
         interne(tnon_sort)
         sortie(X23580)
         valeur(X23555,X23580)
```

II. DRS DU TEXTE PROTOTYPE APRÈS LA RESOLUTION DES ANAPHORES

Seules figurent une partie des DRS ayant été modifées par le module de résolution d'anaphores.

DRS principale du texte

*Le déclenchement d'une règle pragmatique a créé la DRS principale de tout le chapitre (**DRS^34**). Les informations contenues dans le titre y sont insérées, car ce sont les informations principales du texte , l'objet du traitement décrit par le texte. De même le marqueur correspondant à "traitement" a été inséré puisqu'un chapitre correspond à la description d'un traitement. Cela permet de résoudre des références faites au mot "traitement" sans que ce mot n'ait été explicitement mentionné dans le texte. Cette DRS principale constitue, par ailleurs, le contenu du **focus global**.*

---

chap^1:
[drs^34,part^1,part^2,part^3,part^4,part^5]
drs^34:
traitement
dgear_nd
algear_nd
et(algear_nd,dgear_nd)
---

*"... **les** autres **cas** ..."*

drs^8:
cas^1-cas^X9316/X9313
---
not cas^1

*"... **les sorties** sont positionnées..."*

drs^9:
[et(algear_nd,dgear_nd)-sortie^15/X9706,fem,plu,def,X9716,X9718,suj]
id10-event^10/X9729
---
sortie(et(algear_nd,dgear_nd))
positionner(id10,et(algear_nd,dgear_nd), not alarme)

*"Si la vitesse sol est invalide **également** ..."*

drs^15:
condition^10-cas^3/X13197
[vit_sol-X13212^X13213/X13210,fem,sing,X13218,X13220,X13222,suj]
id17-event^17/X13233
[condition^8-condition^X13252/X13249,fem,sing,local,X13259,av,X13263]
---
etre(id17,vit_sol,invalide)-condition^10/X13278
egalement(condition^10,condition^8)
ref_dis(condition^10,condition^8)

"*... **la condition sur la vitesse** est remplie ...*"

drs^16:

id18-event^18/X14269
[inf(vit_conv,vmini)-condition^14/X14285,fem,sing,def,X14295,X14297,suj]
[vit_conv-vitesse^9/X14310,fem,sing,def,X14320,X14322,obj]
---
inf(vit_conv,vmini)-condition^15/X14338
co_ref(inf(vit_conv,vmini),condition^15)
conside2rer(id18,inf(vit_conv,vmini),rempli)
vitesse(vit_conv)
condition(inf(vit_conv,vmini),vit_conv)

*"On positionne **simultanément** l'info interne, Train non sorti, à **la même valeur** que **les deux infos de sorties**."*

drs^31:

id38-event^38/X23913
X23920-event^X23927/X23924
[tnon_sort-info^X23940/X23937,fem,sing,X23945,X23947,X23949,obj]
[X23958-valeur^3/X23962,fem,sing,def,X23972,X23974,obj]
[et(algear_nd,dgear_nd)-sortie^X23993/X23990,X23994,plu,def,2,X24002,obj]
---
positionner(id38,tnon_sort,X23958)
simultanement(id38,X23920)
ref_dis(id38,X23920)
info(tnon_sort)
interne(tnon_sort)
sortie(et(algear_nd,dgear_nd))
valeur(X23958,et(algear_nd,dgear_nd))

III. TRACE DE LA RESOLUTION DES ANAPHORES

Voici une partie de la trace de la résolution des anaphores.

∗ *"... **les** autres **cas** ..."*

*Les sources de contraintes **accessibilité logique** et **focus global** proposent des antécédents. Sur ces candidats chacune des sources de contraintes donne son avis. Un candidat est retenu comme antécédent possible du référent si aucune source ne l'a rejeté.*

[part^2,ph^3,drs^8] ref : X4295-cas^X4302/X4299

Liste proposee par access_log :
[cas^1,tav_ver_b,tga_ver_b,tdr_ver_b,vit_conv,ang_m_ga,ang_m_dr,hrs,algear_nd,dgear_nd]

Liste retenue :[cas^1]

Liste proposee par focus_global :

[[traitement-traitement,mas,sing|X4571],dgear_nd,algear_nd,[et(algear_nd,dgear_nd)-sortie^X4591/X4588,fem,plu,X4596,X4598,X4600,X4602]]

[traitement-traitement,mas,sing|X4571] rejete par type
dgear_nd rejete par type
algear_nd rejete par type
[et(algear_nd,dgear_nd)-sortie^X4591/X4588,fem,plu,X4596,X4598,X4600,X4602] rejete par type

Liste retenue :[cas^1]

antecedent = cas^1

∗ *"... **les sorties** sont positionnées..."*

[part^2,ph^3,drs^9] ref : [X4721-sortie^15/X4725,fem,plu,def,X4735,X4737,suj]

Liste proposee par access_log :
[cas^1,tav_ver_b,tga_ver_b,tdr_ver_b,vit_conv,ang_m_ga,ang_m_dr,hrs,algear_nd,dgear_nd]

cas^1 rejete par type
tav_ver_b rejete par nombre
tga_ver_b rejete par nombre
...
dgear_nd rejete par nombre

Liste proposee par focus_global :
[[traitement-traitement,mas,sing|X5024],dgear_nd,algear_nd,[et(algear_nd,dgear_nd)-sortie^X5044/X5041,fem,plu,X5049,X5051,X5053,X5055]]

[traitement-traitement,mas,sing|X5024] rejete par genre
dgear_nd rejete par nombre
algear_nd rejete par nombre

Liste retenue :[[et(algear_nd,dgear_nd)-sortie^X5044/X5041,fem,plu,X5049,X5051,X5053,X5055]]

antecedent = [et(algear_nd,dgear_nd)-sortie^X4875/X4872,fem,plu,X4880,X4882,X4884,X4886]

∗ *"Si la vitesse sol est invalide **également** ..."*

*"également" met en relation la condition décrite dans la phrase courante (**condition^10** correspondant à "Si la vitesse sol est invalide") avec une condition mentionnée précédemment. L'analyse de l'adverbe "également" a introduit un référent de type **condition**. Seule la source de contraintes **local** proposera des antécédents. Ces antécédents sont pris dans la phrase courante (il faut alors marquer le référent et la **condition^10** comme disjoints pour éviter une auto-référence) ou dans la phrase précédente.*

[part^3,ph^5,drs^15] ref : [X4674-condition^X4594/X4591,fem,sing,local,X4601,av,X4605]

avec les conditions : [],[ref_dis(condition^10,X4674)]

==========

Liste proposee par local :

[condition^10-cas^3/X4732,
[vit_sol-X4747^X4748/X4745,fem,sing,X4753,X4755,X4757,suj],
id17-event^17/X4768,
condition^8-cas^2/X4991,
[vit_conv-X5006^X5007/X5004,fem,sing,X5012,X5014,X5016,suj],
id13-event^13/X5027,id14-event^14/X5099,
[vit_sol-X5114^X5115/X5112,fem,sing,X5120,X5122,X5124,obj],
cas^2]

condition^10-cas^3/X4732 rejete par ref_dis
[vit_sol-X4747^X4748/X4745,fem,sing,X4753,X4755,X4757,suj] rejete par type
id17-event^17/X4768 rejete par type

Liste retenue :[condition^8-cas^2/X4991]

antecedent = condition^8-cas^2/X4991

∗ "*... **la condition sur la vitesse** est remplie ...*"

*Dans le cas présent, il y a recherche groupée d'antécédents pour deux référents (un de type **condition**, l'autre de type **vitesse**) avec la contrainte sémantique que la condition porte sur la vitesse. La source de contraintes **consistance** va donc s'appliquer.*

D'autre part, pour tous les référents de cette partie une règle pragmatique est applicable. Elle indique que les antécédents des référents de la partie CONTROLES-SUR-ALARME sont à rechercher dans la partie REGLE_PRINCIPALE .

*Un nouveau type d'évaluation est illustré dans cet exemple. La source **règle pragmatique** a proposé des antécédents qui ont tous été rejetés. Un deuxième passage est alors lancé. La source **ensemble** est sollicitée pour décomposer les ensembles élément par élément.*

*Finalement deux antécédents ont été retenus: l'un étant le marqueur correspondant à "vitesse conventionnelle inférieure à Vmini" de la partie précédente et proposé par la source **règle pragmatique** ; l'autre étant le marqueur de "si la vitesse sol est invalide également" de la phrase courante et proposé par la source **accessibilité logique** . Il y a ambiguité. L'utilisateur est sollicité pour choisir.*

[part^3,ph^5,drs^16] ref : [X7927-condition^14/X7822,fem,sing,def,X7832,X7834,suj]

avec les conditions : [[X8965-vitesse^9/X8969,fem,sing,def,X8979,X8981,obj]], [condition(X8937,X8965),vitesse(X8965)]

==========

Liste proposee par access_log :

[condition^10-cas^3/X8285,[vit_sol-X8300^X8301/X8298,fem,sing,X8306,X8308,X8310,suj],id17-event^17/X8321,[condition^8-condition^X8340/X8337,fem,sing,local,X8347,av,X8351],id18-event^18/X8450,cas^2,vit_sol,vit_conv]

Liste retenue :[condition^10-cas^3/X8285]

Liste proposee par reg_prag :

[et(condition^7,et(condition^6,et(condition^5,et(condition^4,et(condition^3,et(condition^2,et(condition^1,X8032)))))))-cas^1/X8038,tav_ver_b,tga_ver_b,tdr_ver_b,vit_conv,ang_m_ga,ang_m_dr,hrs,id6-event^6/X8063,[et(algear_nd,dgear_nd)-sortie^15/X8141,fem,plu,def,X8151,X8153,suj],,algear_nd,dgear_nd,id5-event^5/X8106,,id10-event^10/X8164]

et(condition^7,et(condition^6,et(condition^5,et(condition^4,et(condition^3,et(condition^2,et(condition^1,X8032)))))))-cas^1/X8038 rejete par nombre
tav_ver_b rejete par type
tga_ver_b rejete par type
...
hrs rejete par type
id6-event^6/X8063 rejete par type
[et(algear_nd,dgear_nd)-sortie^15/X8141,fem,plu,def,X8151,X8153,suj] rejete par nombre
algear_nd rejete par type
...
id10-event^10/X8164 rejete par type

Liste retenue :[condition^10-cas^3/X7976]

DEUXIEME PASSAGE, Liste proposee par ensemble :

[condition^7,condition^6,condition^5,condition^4,condition^3,condition^2,condition^1,algear_nd,dgear_nd,cas^1]

condition^7 rejete par consistance
condition^6 rejete par consistance
condition^5 rejete par consistance

Liste retenue :[condition^4]

Liste proposee par focus_global :

[[traitement-traitement,mas,sing|X8163],dgear_nd,algear_nd,[et(algear_nd,dgear_nd)-sortie^X8183/X8180,fem,plu,X8188,X8190,X8192,X8194]]

[traitement-traitement,mas,sing|X8163] rejete par genre
dgear_nd rejete par type
algear_nd rejete par type
[et(algear_nd,dgear_nd)-sortie^X8183/X8180,fem,plu,X8188,X8190,X8192,X8194] rejete par nombre

Liste retenue :[condition^10-cas^3/X7976,condition^4]

antecedents = [condition^10-cas^3/X7969,condition^4]

∗ "... *à* ***la*** même ***valeur*** *que* ***les deux infos de sorties*** ..."

Il s'agit ici de trouver la valeur des informations de sortie. Aucune valeur n'étant mentionnée explicitement précédemment, aucun antécédent ne sera trouvé. Pourtant cette référence est légitime puisqu'une information a une et une seule valeur. Cette connaissance est d'ailleurs décrite dans le réseau BACK par les contraintes de cardinalité sur les rôles. Mais l'information correspondante ne peut être pour l'instant exploitée dans ACTES.

part^5,ph^10,drs^31] ref : [X5293-valeur^3/X5200,fem,sing,def,X5210,X5212,obj]

avec les conditions : [[X6080-sortie^X6087/X6084,X6088,plu,def,2,X6096,obj]],[valeur(X6052,X6080)]

==========

Liste proposee par access_log :

[id38-event^38/X5561,X5568-event^X5575/X5572,[tnon_sort-info^X5588/X5585,fem,sing,X5593,X5595,X5597,obj]]

id38-event^38/X5561 rejete par type
[tnon_sort-info^X5588/X5585,fem,sing,X5593,X5595,X5597,obj] rejete par type

---

Liste proposee par focus_global :

[[traitement-traitement,mas,sing|X5476],dgear_nd,algear_nd,[et(algear_nd,dgear_nd)-sortie^X5496/X5493,fem,plu,X5501,X5503,X5505,X5507]]

[traitement-traitement,mas,sing|X5476] rejete par genre
dgear_nd rejete par type
algear_nd rejete par type
[et(algear_nd,dgear_nd)-sortie^X5496/X5493,fem,plu,X5501,X5503,X5505,X5507] rejete par nombre

pas d'antecedent

IV. EXTRAIT DE LA REPRESENTATION DES CONNAISSANCES DU DOMAINE

Nous allons faire quelques remarques concernant les choix que nous avons faits pour représenter les connaissances de notre domaine.

On a décomposé les entrées en booléen et paramètres. En fait, les sorties devraient être décrites de la même façon, mais il se trouve que les sorties que nous avons à traiter sont toujours des booléens; nous ne les avons pas décomposées. Il faudrait faire la distinction entre deux caractéristiques différentes des informations : leur type (entrée/sortie/interne) et leur genre paramètre/booléen).

Un cas est un ensemble de conditions. La relation hiérarchique utilisée n'est pas la relation IS_A.

Nous avons fait la distinction entre les comparaisons (égal/inférieur/supérieur) et ce que nous avons appelé "état". Cette distinction, pertinente dans le cas de l'égalité, correspond en syntaxe aux formes "être" + valeur et "être à" + valeur, respectivement. Dans le second cas, le verbe peut être absent.

Nous avons distingué deux types de verbes : les verbes d'action (forcer, positionner, considérer) et le verbe utiliser. Les premiers lient un objet à un attribut, le second lie deux objets entre eux.

```
/* ensembles d'attributs */
        type_entree = attrset(t_entree).
        type_sortie         = attrset(t_sortie).
        type_interne        = attrset(t_interne).
        tout_type           = attrunion(type_entree,type_sortie,type_interne).

        etat_valide         = attrset(valide).
        etat_invalide       = attrset(invalide).
        etat_validite= attrunion(etat_valide,etat_invalide).
        etat_condition      = attrset(vrai,faux).
        etat        = attrunion(etat_validite,etat_condition).

        entier              = attrset(100,200).
        val_train_ver_bas   = attrset(ver_b,n_ver_b).
        val_alarme          = attrset(alarme,n_alarme).
        ...
        val_vitesse_mini    = attrset(v_mini).
        val_vitesse         = attrunion(entier,val_vitesse_mini).
        val_bool            = attrunion(val_train_ver_bas,etat_validite).
        val_continue        = attrunion(val_hauteur,val_vitesse,val_ang_m).
        val         = attrunion(val_alarme,val_continue,val_bool).


/* le concept chose est le concept racine */
chose = rootconcept.

/* les objets ont des propriétés */
objet = primconcept(specializes(chose),
        restriction(propriete,attribut,[1,in])).

/* les relations font correspondre des objets à des attributs */
relation = primconcept(specializes(chose),
        restriction(relation_objet,objet,[1,in]),
        restriction(relation_attribut,attribut,[1,in])).
```

```
entite = primconcept(specializes(chose)).
event = primconcept(specializes(chose)).

/*types d'informations */
info = primconcept(specializes(objet),
        restriction(a_pour_type,tout_type,[1,1]),
        restriction(a_pour_val,val,[1,1]),
        restriction(a_pour_origine,processeur,[1,2]),
        restriction(a_pour_validite,etat_validite,[1,1]) ).

a_pour_type = primrole(differentiates(propriete)).
a_pour_val = primrole(differentiates(propriete)).

entree = defconcept(specializes(info),
        value_restriction(a_pour_type,type_entree)).
sortie = defconcept(specializes(info),
        value_restriction(a_pour_type,type_sortie),
        value_restriction(a_pour_val,val_alarme)).
param = defconcept(specializes(entree),
        value_restriction(a_pour_val,val_continue)).
bool = defconcept(specializes(entree),
        value_restriction(a_pour_val,val_bool)).
n_redond1 = primconcept(specializes(info),
        restriction(a_pour_origine,bisa_1,[1,1])).
redond = primconcept(specializes(info),
        restriction(a_pour_origine,processeur,[2,2])).

/* entités */
processeur = primconcept(specializes(entite)).
bisa_1 = primconcept(specializes(processeur)).
        individual(bisa_1).
btm_2 = primconcept(specializes(processeur)).
        individual(btm_2).

/* ====== LES INFOS D'ENTREES ======*/
/* les trains */
info_train = primconcept(specializes(redond)).
t_ver_b = primconcept(specializes(bool),
        specializes(info_train),
        value_restriction(a_pour_val,val_train_ver_bas)).
tav_ver_b = primconcept(specializes(t_ver_b)).
        individual(tav_ver_b).
vt_redond = primconcept(specializes(bool),
        value_restriction(a_pour_val,etat_validite)).
        individual(vt_redond).

/* les vitesses */
vitesse = primconcept(specializes(param),
        value_restriction(a_pour_val,val_vitesse) ).
vit_conv = primconcept(specializes(vitesse),
        specializes(n_redond1)).
        individual(vit_conv).
vit_sol = primconcept(specializes(vitesse),
        specializes(n_redond1)).
        individual(vit_sol).

/* ====== LES INFOS DE SORTIE ======*/
```

```
algear_nd = primconcept(specializes(sortie) ).
        individual(algear_nd).

/* ====== CONDITIONS ET ACTIONS ====== */

/* un cas est un ensemble de conditions */
cas = primconcept(specializes(relation)).
condition = primconcept(specializes(cas),
        restriction(relation_objet,info,[1,1])).

/* comparaisons */
compar = primconcept(specializes(condition),
        restriction(relation_attribut,val_continue,[1,1])).
eg = primconcept(specializes(compar)).
inf = primconcept(specializes(compar)).

/* états */
etre = primconcept(specializes(condition),
        restriction(relation_attribut,etat,[1,1])).

/* actions */
action = primconcept(specializes(relation)
        restriction(action_event,event,[1,1])).
forcer = primconcept(specializes(action),
        restriction(relation_objet,sortie,[1,1]),
        restriction(relation_attribut,val,[1,1])).
positionner = primconcept(specializes(action),
        restriction(relation_objet,sortie,[1,1]),
        restriction(relation_attribut,val,[1,1])).
considerer = primconcept(specializes(action),
        restriction(relation_attribut,etat,[1,1])).
utiliser = primconcept(specializes(chose),
        restriction(utiliser_condition,condition,[1,1]),
        restriction(utiliser_info,info,[1,in])).
```

V. EXEMPLE DE GRAMMAIRE AVAG

TYPAGE ET UNIFICATION ETENDUE
le fichier de representation des connaissances
en BACK s'appelle 'domaine/rc_exemp8'
en format ACTES 'domaine/core_exemp8'
c'est ce dernier qu'il faut charger
(menu BACK)

EXEMPLES D'ANALYSE :
'un homme dort' (reconnu au niveau de 'ph')
'les hommes dorment'
'un homme qui dort' (reconnu au niveau de 'gn')
'un astronome observe une planete avec une lunette'
(deux analyses)
'un astronome observe avec une lunette une planete'
NON RECONNUS:contraintes de type non respectees
'une lunette dort'
'une planete observe'
'un astronome observe un homme avec une planete'
*/

/* ----- les gabarits ------ */

soit fs:
<head accord genre> = fem
<head accord nbre> = sing.
soit ms:
<head accord genre> = mas
<head accord nbre> = sing.

/* ------ lexique --------*/

un:
<cat> = art
ms.

une:
<cat> = art
fs.

les :
<cat> = art
<head accord nbre> = plu
<head accord genre> = { mas, fem }.

astronome :
<cat> = nom
ms
<head sem arg1> = X1:astronome
<head sem pred> = astronome(X1).

homme:
^astronome
<head sem arg1> = X1:homme
<head sem pred> = homme(X1).

```
lunette :
        <cat> = nom
         fs
        <head sem arg1> = X1:instrument
        <head sem pred> = lunette(X1).

planete:
          ^ lunette
        <head sem arg1> = X1:objet
        <head sem pred> = planete(X1).

astronomes:
        ^astronome
        <head  accord nbre> = plu.

lunettes :
        ^lunette
        <head  accord nbre> = plu.

planetes :
        ^planete
        <head  accord nbre> = plu.

dort:
         <cat> = verb
         <head accord nbre> = sing
         <sousclasse> = intrans
         <head sem arg1> = X1:humain
         <head sem pred> = dort(X1).

dorment:
         ^dort
         <head accord nbre> = plu.

observe:
          ^dort
         <sousclasse>= trans
         <head sem arg1> = X1:humain
         <head sem arg3> = X3:instrument
         <head sem pred> = observe(X1,X2,X3).

observent:
          ^observe
         <head accord nbre> = plu.

avec:
          <cat> = prep.

/* -----  le groupe nominal simple. ----- */
gn -> art nom gp
        <art head  accord> = <nom head  accord>
        <gn  head accord> = <nom head accord>
        <gn  head sem arg1> = <nom head sem arg1>.

gn -> art nom [qui] {!} ph
       <art head accord> = <nom head accord>
        <nom head > = <ph slash>
        <gn head > = <nom head>.
```

```
gn -> art nom {!}
        <art head  accord> = <nom head  accord>
        <gn  head > = <nom head>.

gn ->
        <gn head> # X0.

ph -> gn gv
        <ph slash> = <gn head>
        <gn head accord> = <gv head accord>
        <gn head sem arg1> :=: <gv head sem arg1>.

gv -> verb gn
        <verb sousclasse> = trans
        <gv head> = <verb head>
        <verb head sem arg2> :=: <gn head sem arg1>.

gv -> verb, gn, gp  [verb<gn  verb<gp]
        <verb sousclasse> = trans
        <gv head> = <verb head>
        <verb head sem arg2> :=: <gn head sem arg1>
        <verb head sem arg3> :=: <gp head sem arg1>.

gv -> verb
        <gv head> = <verb head>.

gp -> prep gn
        <gp head> = <gn head>.
```

/* REPRESENTATION DES CONNAISSANCES */

```
chose       = rootconcept.

objet       = primconcept( specializes(chose)).
planete     = primconcept( specializes(objet)).
instrument  = primconcept( specializes(objet)).
lunette     = primconcept( specializes(instrument)).

humain      = primconcept( specializes(chose)).
homme       = primconcept( specializes(humain)).
astronome   = primconcept( specializes(humain)).

action      = primconcept( specializes(chose),
                    value_restriction(agent,humain),
                    value_restriction(theme,chose)).
observe     = primconcept( specializes(action),
                    value_restriction(moyen,instrument)).
```